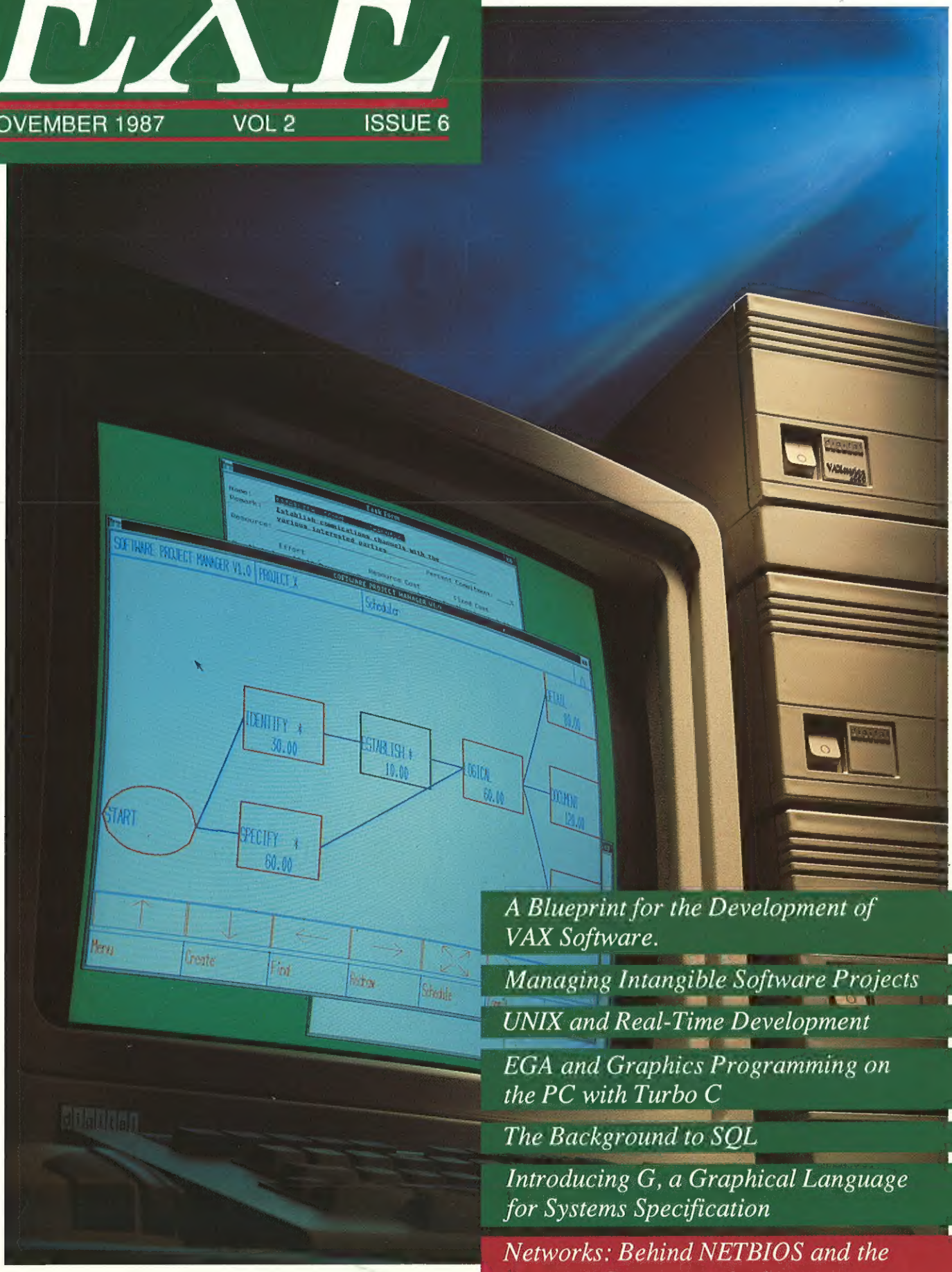
.EXE

NOVEMBER 1987 VOL 2 ISSUE 6

The Software Developers' Magazine
Software • Hard Facts



A Blueprint for the Development of VAX Software.

Managing Intangible Software Projects

UNIX and Real-Time Development

EGA and Graphics Programming on the PC with Turbo C

The Background to SQL

Introducing G, a Graphical Language for Systems Specification

Networks: Behind NETBIOS and the Network Computing Architecture.

REACH FOR THE SKY.

THE MOST PROGRAMMABLE APPLICATIONS DEVELOPMENT SYSTEM IN THE WORLD

For the professional software writer SkyBase 4 is the perfect tool for developing complete business applications.

SkyBase 4 Features:
Windows · Scrolling screens
Tree structured files · Simple language
Flexible reports · Data dictionary
Help facility · Cross machine development capability · SkyMaster 4 accounting system link

With these unique features, you can create single or multi-user software quickly and efficiently to run under Xenix, Unix or DOS for almost any business application you can think of.

To us this sounds like THE definition of a 4th generation language.

To find out more, phone

(0527) 36299
And Reach For The Sky.

SKY SOFTWARE

Sky Software Ltd, 13 New Road, Bromsgrove, Worcs B60 2JG

CIRCLE NO 781

CONTENTS

Editor: Ian Adams
Deputy Editor: Robert Schifreen
News Editors: Nick Roach, Hobbit Coward
Advertising Manager: Jonathon Howell
Advertising Executive Helena Adams
Design: Katey M.
Print: Chase Web Offset

All Change: Continued developments at .EXE include the appointment of a Deputy Editor (read his inaugural EDLIN this month) and a massive expansion to 80 pages for this issue, much to the annoyance of our typesetting and proof-reading cronies (Hotzenplotz, et al) who were just getting comfortable with 64 pages.

Editorial: We welcome articles from prospective authors and invite submissions from readers. Please write or phone for our contributor's notes and rates of payment. The aim of .EXE's Editorial Board is to meet, reflect and advise on the direction of the software industry. Members of the board are contributors or vendors' representatives.

Subscriptions: .EXE is available only on subscription. An annual subscription for 10 issues costs £35 in the UK and £45 for overseas.

.EXE is published each month excluding August and December. It is an independent publication and is not affiliated in any way to any manufacturer or vendor of hardware, software or services. .EXE Magazine is published by Process Communications Ltd, 10 Barley Mow Passage, Chiswick, London W4 4PH, England. Tel: 01 994 6477. ISSN 0268-6872.

COVER FEATURE

CASE

UNIX and REAL-TIME

NETWORKING

PC GRAPHICS

REGULARS

EDLIN

If I tell you mine, will you tell me yours...?

.EXE's Deputy Editor, Robert Schifreen, is to take the editorial reins in January 1987. This is his inaugural EDLIN.

I didn't know exactly what the thing was, but it made enough noise to attract my attention. A week or so later, I dared to ask what it was. Officially, it was 'the school computer'. Actually, it was only a terminal, and 75 baud at that. It was linked to an ICL 1900 at Hendon Town Hall, London, and ran something called Maximop. 'Don't bother learning about Maximop', I was told, "cos George III is on the way.' It never came.

Developing software was slow. A couple of sixth-form boys were preparing their project for O-level computer science and, if they wanted a listing of their program, they'd set the thing going before morning assembly and, if all was well, it would be ready by lunchtime. If they wanted the listing urgently, they had to ASSI CON,RP or something, and the remote lineprinter at the Town Hall would churn it out and the teacher would whizz round on his bike to collect it. (Ah, the days of dumping the system's entire core to the remote printer and getting nasty phone calls from the town hall when the borough's paper stock ran out!)

In early 1982 the school got its first micro – an RML 380Z with, wonder of wonders, a real floppy disk drive. Each disk held a whole 72K, so with CP/M and the BASIC interpreter on there, you could even fit a couple of BASIC programs as long as they weren't too long.

Having sold enough cameras on enough Saturdays, I could afford a computer of my own. I eventually ended up with a Superboard II, Spectrum and QL. I taught myself BASIC and 6502 machine code. Over a period of 4 months, I wrote a machine code monitor for the Superboard, that revectored the keyboard input and added a dozen extra functions to the machine. Progress was slow, as the machine's 8K memory (it came with 4K but I saved up for the expansion) had no room for the assembler so I had to enter everything by hand, in Hex. I also had no printer, so listings were hand-written by disassembling (again, from the hex).

In April 1983, I got my first full time job. I was still a teenager – just – and was employed as Reader Services Administrator at Computer And Video Games magazine. This involved answering the phone and, before the caller could finish saying 'that game you published doesn't run – I keep getting Variable Not Found At Line 11780' I'd butt in with 'I can assure you there's nothing wrong with the game. I checked it myself'.

After six months or so, they offered to let me try my hand at writing. I knew then that I was totally hooked on Journalism as well as computing in general. The two seemed to go together wonderfully. I don't suppose I really knew exactly how addicted I was until, after 19 months at C&VG, I went to Epson to write manuals. I realised, from day one I suppose, that I wanted to be back in journalism, but Epson taught me a lot about PCs, MS-DOS and printers during my stay and I value the time greatly.

Having tired of Epson, I freelanced for a few months. This involved staying in bed, mostly, but getting up in plenty of time to go to product launches at posh London hotels. I used to have a badge that said 'Robert Schifreen – Freelunch Journalist'. In November 1986 I joined PCW – sorry Personal Computer World – as a staff writer. I was back in full time journalism and the rest, as they say... well you know what they say.

I suppose I've been pretty lucky, though the odd dollop of publicity helped, too. At 10.06pm on Tuesday, 29th March 1985 I became the first person in Britain to be arrested for hacking. When I had typed in Prince Philip's password, said the Detective Inspector, this was equivalent to forging his signature. A year later a jury decided this to be the case and I was fined. (The journalist from The Sun was not present when the verdict was announced. By day 3, he had turned to another Fleet Street colleague and said 'Sod this. If there's no sex angle to Prestel I'm off). Another year, and The Lord Chief Justice overturned the lower court, so I'm currently innocent. The House of Lords will probably make the final decision some time in 1988 or 1989.

So, then. I've told you some of my secrets, now it's only fair for you to tell me something in return. I know what I'd like to see in .EXE magazine, and will be spending this Christmas planning the next 6 issues. However, a magazine exists to serve its readers, so please write and tell me what you want to see. More coverage of other languages, like Forth, Occam, Lisp? Technical features on MS-DOS and BIOS functions? Programming for Windows and OS/2? Stuff about parallel processing and supercomputers? 4GL's? Book reviews? Hardware reviews? These are some of my current thoughts, but there's still plenty of blank pages for your ideas. Contact me at the editorial office. Alternatively, I'm on Telecom Gold 72:MAG90141 or Prestel 219997602. I look forward to working with you.

BASIC LANGUAGE

New Quickbasic V3.0 uprates the competition with TURBO-BASIC.

BASIC INTERPRETERS

BBC Basic (86)	PC-DOS	£ 95
Professional BASIC	PC-DOS	£ 70
TrueBasic v2.0	PC-DOS	£ 95
Microsoft MS-BASIC	MS-DOS	£210
MEGABASIC v5.2	MS-DOS	£235
Dig.Res. CBASIC	CP/M-86	£290
MEGABASIC	CP/M-86	£235
MEGABASIC	MP/M-86	£365
BBC BASIC	Z80+CP/M-80	£ 95
Dig.Res. CBASIC	CP/M-80	£130
Microsoft MBASIC	CP/M-80	£ 60
MEGABASIC	CP/M-80	£195

BASIC COMPILERS

Microsoft QuickBASIC	PC-DOS	£ 60
Softaid MTBASIC	PC-DOS	£ 60
Turbo Basic	PC-DOS	£ 60
ZBASIC	PC-DOS	£ 75
Alcor Multi-Basic	MS-DOS	£ 75
Microsoft MS-BASIC	MS-DOS	£235
Dig.Res. CBASIC	MS-DOS	£390
Dig.Res. CBASIC	CP/M-86	£390
Dig.Res. CBASIC	CP/M-80	£435
ZBASIC	Z80+CP/M-80	£ 75
Softaid MTBASIC	Z80+CP/M-80	£ 60
Alcor Multi-Basic	Z80+CP/M-80	£ 85

LIBRARIES & UTILITIES

Database

CADSAM (source code)	MS-DOS	£ 75
Btrieve	MS-BASIC + MS-DOS	£180
Btrieve/N	MS-BASIC + MS-DOS	£430
Multikey	MS-BASIC + MS-DOS	£145
CADSAM (source code)	CP/M-80	£ 75

Graphics

Halo	MS-BASIC + MS-DOS	£175
GSS Graph.Dev.Tkt	PC-DOS	£335

Sundries

Finally Quickbasic +	PC-DOS	£ 75
PANEL Screen Manager	MS-DOS	£100
Wiley Scientific Lib.	PC-DOS	£110

Tuning & Debugging

Betatools Dev.System	PC-DOS	£100
Vicar	MS-DOS	£ 35

FORTRAN COMPILERS

New V4.00 from Microsoft now mean four excellent 77's to choose from.

Lahey F77L v2.20	MS-DOS	£360
RM/FORTRAN 77 v2.11	MS-DOS	£400
MS-FORTRAN 77 v4	MS-DOS	£250
Pro Fortran 77 v1.22	MS-DOS	£220
Pro Fortran v2.144	MS-DOS	£220
Utah Fortran	MS-DOS	£ 35
FS-Fort.(GGA & Herc)	MS-DOS	£ 35
Pro Fortran v2.1	CP/M-86	£220
Pro Fortran v1.25	CP/M-80	£220
Nevada Fortran v3.3	CP/M-80	£ 35
Pro Fortran 77	ATARI 520ST	£120

We have Fortran Libraries in stock.

PASCAL LANGUAGE

Metaware excellence now on Concurrent Dos 286 and on the 386. Better Turbo compatibility on new version Pro Pas

ALICE Pascal Intrprtr.	PC-DOS	£ 80
Dr Pascal Interpreter	MS-DOS	£ 39
Marshall Pascal v2.01	MS-DOS	£150
Metaware Prof.Pascal	MS-DOS	£415
Metaware Prof.Pas/386	MS-DOS	£610
Microsoft Pascal v3.32	MS-DOS	£180
Mystic Pascal v1.6E	PC-DOS	£ 29
Oregon Pascal-2	MS-DOS	£280
Prospero Pascal v3.0	MS-DOS	£220
Turbo-Pascal MS-DOS &	PC-DOS	£ 60
Pecan P-Sys w.UCSD Pas.	IBM-PC	£ 80
Pecan P-Sys Pascal Prof.	IBM-PC	£155
Metaware Prof.Pascal	C-DOS	£415
Pro-Pascal v2.14	CP/M-86	£220
Turbo-Pascal v3.01	CP/M-86	£ 60
Pascal MT+ v5.6	CP/M-80	£ 99
Pascal MT+ v5.6.1	CP/M-80	£290
Pro-Pascal v2.18	CP/M-80	£220
Turbo-Pascal v3.01	CP/M-80	£ 45
MCC Pascal	ATARI ST	£ 75
Pecan P-Sys w.UCSD Pas	ATARI ST	£ 80
Pecan P-Sys w.UCSD Pas	APPLE][	£ 80

We have many Pascal Libraries. Enquire

PROGRAMMING TOOLS

Ada Compilers	Algol Compilers
Assemblers	Assembler Libs.
Basic Compilers	Basic Interpreters
Basic Utilities	Basic Libraries
BCPL Compilers	C Compilers
C Interpreters	C Libraries
C Utilities	Cobol Compilers
Comms.Libraries	Cross Assemblers
Database Libs.	Debuggers
Dis-assemblers	Editors
Engineers Libs.	Expert Systems
Forth	Fortran Compilers
Fortran Libraries	Graphics Libraries
Icon	Linkers
Lisp	Modula-2
Nial Interpreters	OPS 5
Pascal Compilers	Pascal Libraries
Prolog	Rexx
Screen Libraries	Smalltalk
Snobol	

We stock many items for which there is no space in these advertisements.

CROSS ASSEMBLERS

We supply cross-assemblers by Avocet, 2500AD, IAR Systems and Pecan hosted on MS-DOS, CP/M-86 and CP/M-80 with over 30 target processors.
In total over 200 products with no space to list them here.
We hold some stock but you should allow 10-14 days for delivery.
Please call for information or advice.

ASSEMBLERS

Microsofts'Macro-86 V5.00 with Codeview may be here by now.

2500AD 8086 Asm.	MS-DOS	£ 80
Dig.Res. RASM-86	MS-DOS	£180
MS Macro-86 v4.0	MS-DOS	£ 90
Phoenix PASM-86	MS-DOS	£115
Phar Lap 386	MS-DOS	£415
Visible Computer 8088	PC-DOS	£ 60
2500AD 8086 Asm.	CP/M-86	£ 80
Dig.Res. RASM-86	CP/M-86	£180
2500AD Z80 ASM	CP/M-80	£ 80
Dig.Res. RMAC	CP/M-80	£180
Microsoft Macro-80	CP/M-80	£ 60
SLR Z80ASM	CP/M-80	£ 45
SLR Z80ASM-PLUS	CP/M-80	£175
SLR MAC	CP/M-80	£ 45
SLR MAC-PLUS	CP/M-80	£175
SLR 180 (Hitachi)	CP/M-80	£ 45
SLR 180-PLUS (Hitachi)	CP/M-80	£175

Not all assemblers are supplied with a linker. Check before ordering.

PASCAL LIBRARIES

TURBO PASCAL LIBRARIES

Blaise Power Tools Plus	PC-DOS	£ 70
Blaise Turbo Asyn.Plus	PC-DOS	£ 70
Mathpak 87 (TP)	MS-DOS	£ 60
Paragon Supertools	PC-DOS	£ 65
RM Graph Nimbus +	MS-DOS	£ 49
Science & Eng.Tools	MS-DOS	£ 50
Report Builder	MS-DOS	£ 70
System Builder	MS-DOS	£ 90
Turbo Halo Univ.Graph.	PC-DOS	£105
T-Debug Plus v2	PC-DOS	£ 35
Turbo Database CP/M &	MS-DOS	£ 45
Turbo Editor Toolbox	PC-DOS	£ 45
Turbo Extender	PC-DOS	£ 55
Turbo Gameworks	PC-DOS	£ 35
Turbo Graphix Toolbox	PC-DOS	£ 49
Turbo Advantage (Lader)	MS-DOS	£ 60
Turbolink Plus v3.15A	PC-DOS	£ 60
Turbopower Utilities	PC-DOS	£ 60
Turbo Optimiser	PC-DOS	£ 45
Turbo Professional	PC-DOS	£ 45
Turbo Screen CP/M,MS,	PC-DOS	£ 60
Turbo Tutor CP/M &	MS-DOS	£ 29
TurboWINDOWS/Pasc.(TP)	PC-DOS	£ 55

GENERAL PASCAL LIBRARIES

Blaise Tools (s'ce) (MS)	PC-DOS	£ 85
Blaise Tools 2 (s'ce)	PC-DOS	£ 80
Blaise Asynch (s'ce MS)	PC-DOS	£120
Btrieve (MS)	PC-DOS	£180
MetaWINDOWS (MS)	PC-DOS	£110
Halo (MS)	PC-DOS	£175
Blaise View Mngr. (MS)	PC-DOS	£205
Shark database (Propas)	MS-DOS	£250
Prospect Graphics (Pro)	MS-DOS	£ 80
Panel (Screen) (MS)	MS-DOS	£205
Shark database (Propas)	CP/M-86	£250
Prospect Graphics (Pro)	CP/M-86	£ 80
Shark database (Propas)	CP/M-80	£150

DISK COPYING SERVICE

We can copy files to and from 400 disk formats including CP/M, CP/M-86, MS- DOS, PC-DOS, ISIS, APPLE, SIRIUS, BBC, TORCH, APRICOT, HP150, TRSDOS, DEC RT- 11, IBM BEF, ATARI520, AMSTRAD.

Our charge is £10.00 + disk + VAT with discounts on small quantities and disks are normally despatched within 24hrs of receipt.

For more information call us.

MODULA -2 COMPILERS

New Farbware compiler. FTL is an excellent value learning tool .

Pecan P-Sys. w.Mod-2	PC-DOS	£ 80
Farbware Modula-2	MS-DOS	£ 70
FTL Modula-2 (sml.mem)	MS-DOS	£ 45
Interface M2-SDS	PC-DOS	£ 75
Interface M2-SDS-XP	PC-DOS	£185
Modula-2/86 Apprentice	PC-DOS	£ 70
Modula-2/86 Wizard	PC-DOS	£140
Modula Corp.PC Mod.2	PC-DOS	£150
Modula 2/86	CP/M-86	£410
FTL Modula-2	Z80/CP/M-80	£ 45
Hochstrasser Mod.2	Z80/CP/M-80	£100
Turbo Modula-2	Z80/CP/M-80	£ 55
TDI Modula-2	ATARI 520ST	£ 75
MacModula-2	MACINTOSH	£100

PRICES & DELIVERY

Prices do not include VAT or other local taxes but do include delivery in UK and Europe.

Please check prices at time of order, ads are prepared some weeks before publication.

This page lists some of our products. Call us for a complete pricelist.

Order by phone with your credit card.

GREY MATTER
4 Prigg Meadow, Ashburton, Devon TQ13 7DF.
TEL. (0364) 53499

GREY MATTER
4 Prigg Meadow, Ashburton, Devon TQ13 7DF.
TEL. (0364) 53499

GREY MATTER
4 Prigg Meadow, Ashburton, Devon TQ13 7DF.
TEL. (0364) 53499

Managing 'Intangible' Projects

It may seem clear how a project management tool can help in building the M25, but very unclear how the same principle can apply to 'intangible' software projects. Anne Neale believes there is really little difference and introduces the IPSE concept.

Imagine being faced with the task of constructing a chemical plant. You are not told what chemicals the plant will be required to manufacture, or in what quantities, or even when. Nor are you given any specify information about the kinds of processes you will need to use to build the plant.

Sounds crazy. Yet stripped down to their basics some large software projects start off in much the same haphazard way. Even though the computer business has matured and many more users have gained a much deeper understanding of computer use, some of the fundamental problems of big project management still remain. If anything they have become more complex as the options in software technology have proliferated.

On the face of it, a major software project seems like any other big engineering task – building an oil platform, constructing the M25, digging the Channel Tunnel. The project will be carried out over a lengthy period of time, using a range of complementary technologies and teams of people versed in different skills.

Yet where software is concerned, first impressions can be deceptive. For the oil platform, M25 and Channel Tunnel are projects in which the parameters and resources are clearly defined. Software is not like that. Over the years the computer industry's philosophers have wrestled with definitions of software, yet these speculations are germane to the central problem of managing large scale software projects. Quite simply, software is an idea. And like all ideas it is difficult to define with scientific precision.

Yet the problem is even more complex. For inevitably a major software project starts out not as one person's but as many people's ideas. Synthesising those ideas into a specification is the first stage of a software project, and it is often done far from well. As a result, in not a few projects, the software development sets out with its route not clearly marked. According to the old Chinese proverb, a journey of a thousand miles starts with a single step. The problem with many software projects is that not all of them turn out to be in the same direction.

The difficulty in many software projects is not just that software is intangible, but that its intangibility has not even been clearly defined.

Classic Projects

Too often, the project will start ill-defined, but then that inadequate 'definition' will be changed as the project progresses. Here, another parallel with M25s and Channel Tunnels illuminates a distinctive problem in software development – the relationship between project defining management and project managing technicians. It is always possible to change the route of the M25 when it is still only a line on a map. Yet after the bulldozers have moved in it is only too obvious that it is too late to change.

Software projects do not benefit from that kind of physical certainty. Because of their intangible nature, end users can never actually see what progress has been made in construction – although fourth generation languages and software prototyping are beginning to change that. Even then, the user tends to see only individual screens and not the inter-relationships that are the important factors in a large integrated software system. The user, therefore, feels there are less constraints to changing specifications as a project progresses. Yet in software terms, some changes can be as devastating as digging up several miles on concreted road and re-laying them a few miles away.

There are, of course, other problems of project management. The project will harness the talents of many skilled staff, project teams will operate from different sites, and there will be a multitude of inter-related tasks to be completed on schedule.

News about the Microsoft Languages Family

Optimizing Your Programs with the Microsoft® C Optimizing Compiler Version 5.0

Fast execution speed is the single most important feature of a C compiler. Volume 2, Number 2 of the Microsoft Languages Newsletter talked about the optimizations available in Microsoft C Version 4.0. Microsoft C Version 5.0 takes these optimizations further. For example,

```
for (i = 0; i < 25; i++)          becomes          tmp = a*b;
        array[i] = a*b;                            for (i = 0; i < 25; i++)
                                                   array [i] = tmp;
```

Since a and b are not affected by the loop, they are moved outside of the loop. This optimization is called *invariant code motion.* The Microsoft C Optimizing Compiler also uses instructions available on the 8086 to optimize specialized loops. Initialization and memory movement loops are two examples. The optimizer generates REP STOSW and REP MOVSW instructions for

```
int i, x[25];                     and              int i, x[25], y[25];
for (i = 0; i < 25; i++)                           for (i = 0; i < 25; i++)
        x[i] = 0;                                          x[i] = y[i];
```

The following example is more complicated. The optimizer rewrites array references as pointer references because they are more efficient.

```
int i, x[25];                     becomes          int i, x[25], *ptr;
for (i = 0; i < 25; i++)                           for (i = 0, ptr = x; i< 25; i++, ptr++)
        x[i] = i*4;                                        *ptr = i*4;
```

Then the optimizer puts key variables in registers using *loop enregistering* and changes the loop incrementation using a process called *strength reduction.* The loop becomes

```
int i, x[25];
i = 25;
{
        register int j;
        register int *ptr;
        for (j = 0, ptr = x; j < 100; j + = 4, ptr++)
                *ptr = j;
}
```

The final form of the loop uses registers for key values and exchanges addition instructions for multiplication instructions. Here is the output of the Microsoft C Optimizing Compiler in 8086 assembly code.

```
        mov     WORD PTR [bp-52], 25        ; set final value of i to 25
        mov     di,bp
        sub     di,50                       ; load pointer to x
        sub     si,si                       ; set temporary register variable to 0
                                            ; this variable is used as the loop counter
$L20000:
        mov     WORD PTR [di], si           ; set the array value
        add     di,2                        ; increment pointer by 2
        add     si,4                        ; increment loop counter by 4
        cmp     si,96                       ; check if we are at the end of the loop
        jle     $L20000
```

What is the result of these optimizations? Programs compiled with Microsoft C Version 5.0 run 15 to 30 percent faster than those compiled with Version 4.0.

For more information on the products and features discussed in the Newsletter,

Write to: Microsoft Language Newsletter

49 de Montfort Road

Reading Berks RG1 8LP

Or Phone : 0734 500741

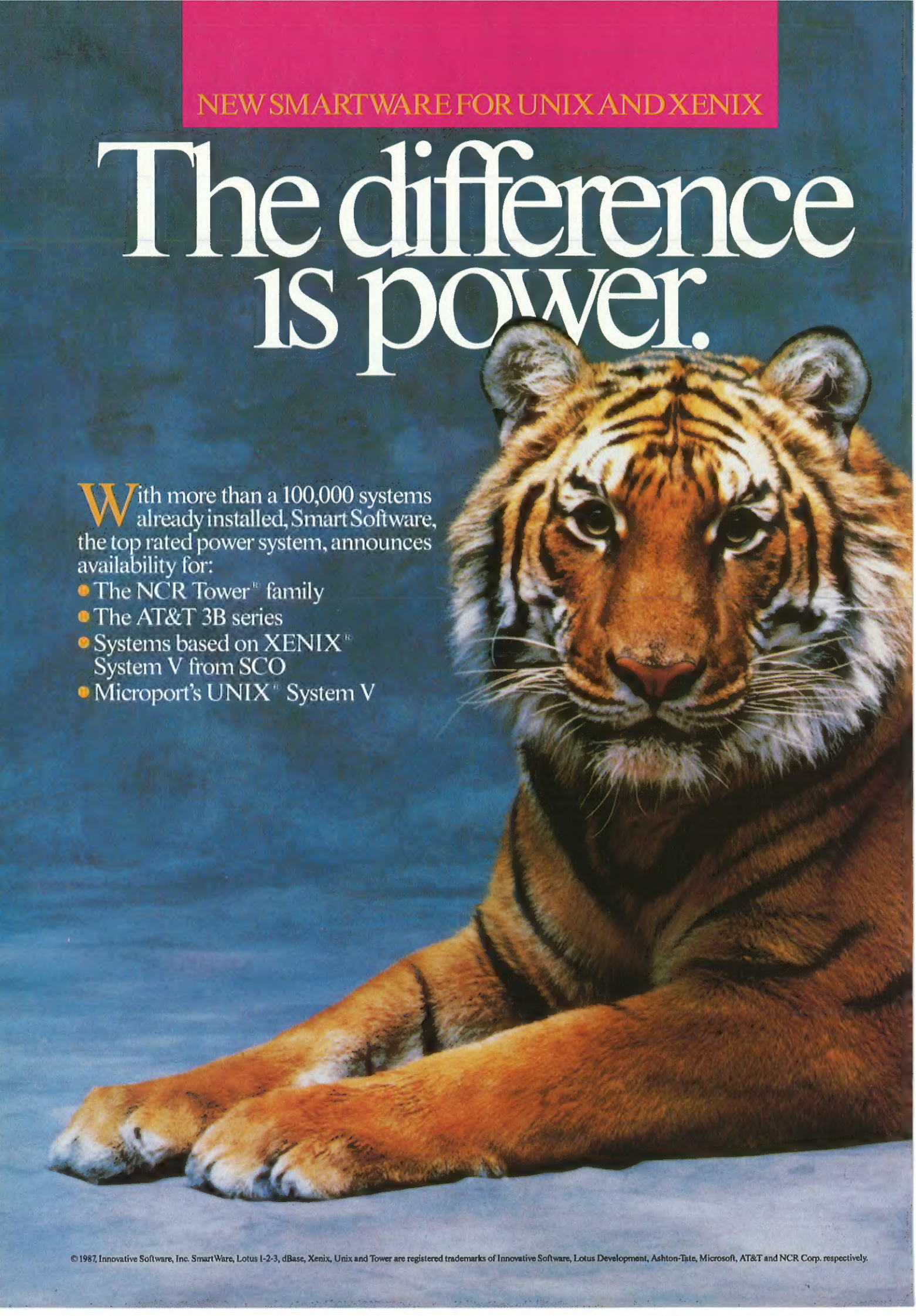
Latest DOS Versions:	
Microsoft C Compiler	5.00
Microsoft QuickC	1.00
Microsoft COBOL	2.20
Microsoft FORTRAN	4.01
Microsoft Macro Assembler	5.00
Microsoft Pascal	3.32
Microsoft QuickBASIC	3.00

Look for the Microsoft Languages Newletter every month in .EXE.

Microsoft®

NEW SMARTWARE FOR UNIX AND XENIX

The difference is power.



With more than a 100,000 systems already installed, Smart Software, the top rated power system, announces availability for:

- The NCR Tower® family
- The AT&T 3B series
- Systems based on XENIX® System V from SCO
- Microport's UNIX® System V

Power to Every User

No other UNIX™ or XENIX™ software has the power to do all the things Smart does. Every user has access to any part of Smart's completely integrated *office automation* solution: word processing, database, spreadsheet, business graphics, calendaring and communications.

Custom Applications

The Smart Application Programming Language combines natural language syntax with the unique ability to program dynamically across all the modules. And Smart's library of over 100 built-in business functions greatly reduces valuable programming time.

Smart "READ" command makes it easy to import data from other file formats.

Smart "SEND" command makes it easy to move data between applications.

Exploit IBM PC Data

Now with Smart, you can read and write data and formulas created in Lotus 1-2-3 or stored in dBase III for use in your UNIX system.

DOS/UNIX Connection

With Smart you can network DOS PCs and UNIX workstations with completely transparent data exchange. Which, according to Information Week, makes Smart "the first to provide DOS-UNIX-LAN connectivity without sacrificing power and features."

But why not see for yourself?

Just return the coupon below, or call Innovative Software on **01-890 8641.** We'll rush you by return your free Smart demo disk and information kit.

SmartWare®
from Innovative Software

FREE SMART DEMO DISK

E EX11

Please send by return the free (tick) ☐ Xenix demo disk ☐ DOS demo disk.
TICK YOUR STATUS ☐ individual computer user ☐ work in a business with many computers
☐ computer store retailer ☐ VAR, VAD, consultant or developer

name _____ position _____

organisation _____

address _____

_____

telephone _____

Complete and return this coupon to: Innovative Software International Limited, Lenexa House, Plane Tree Crescent, Feltham, Middlesex.

CIRCLE NO 784

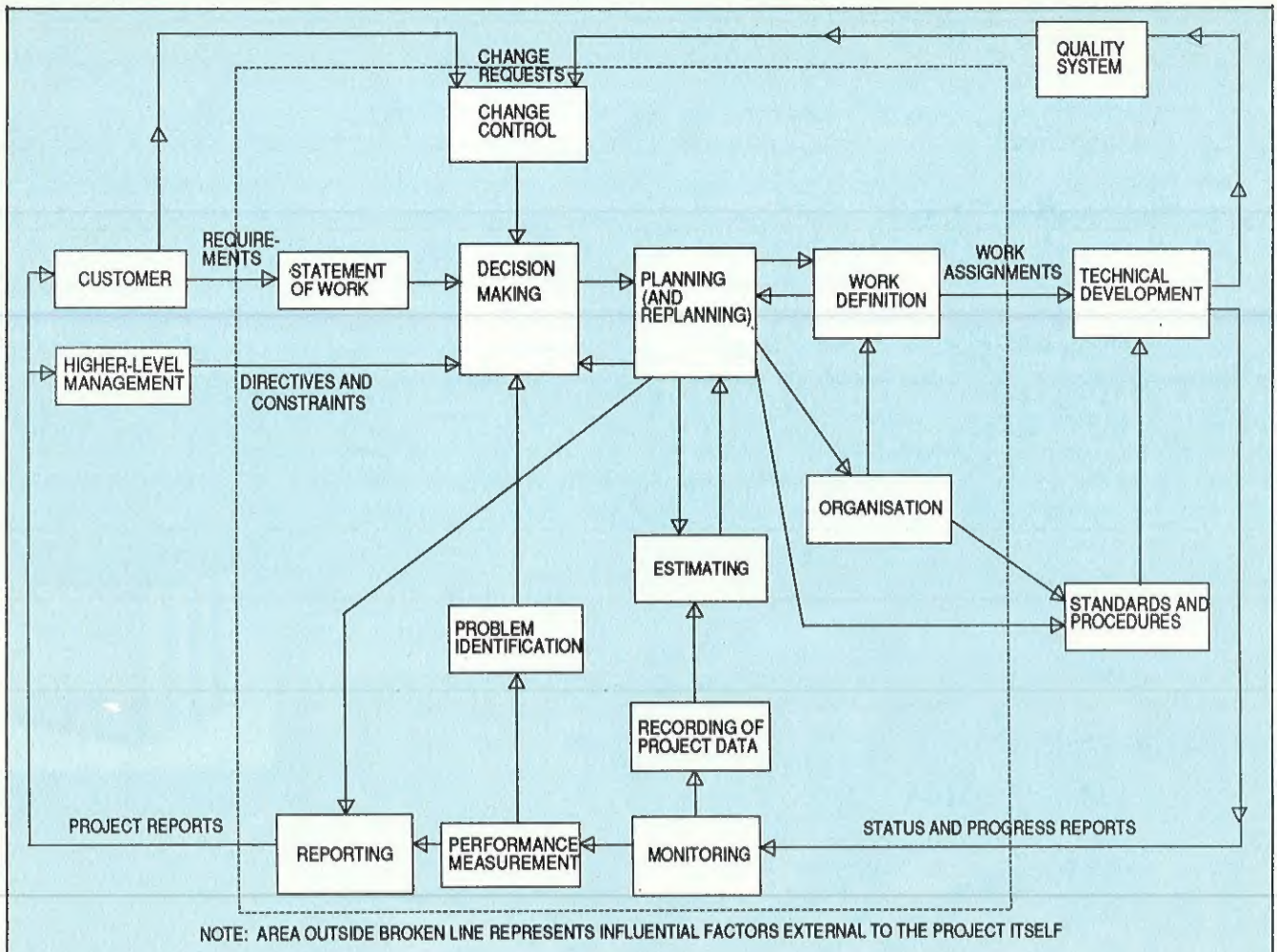


Figure 1: The project management process.

There are also other problems. A large software project may well use a variety of program development tools and methodologies. There will be complex data management needs. The project may harness many workstations and many different host systems. These problems, while requiring skilled project management, are not unique to software projects. They crop up in many other areas of engineering. It is the application of those factors to a highly intangible project, however, that is the central problem. It seems clear that a major software development project can benefit from the application of tried and tested project management methodologies – especially when applied in a computerised form.

The central question lies in how to make those management techniques work best on an intangible project.

Management Techniques

The seeds of the answer lie in the evolution of project management techniques. Guns were thundering over the Somme when Henry L Gantt, an American pioneer of scientific management, developed a graphical representation of the stages needed to complete a project by a target date. He lent his name to a chart consisting of bars representing activities of specific duration that have to take place at specific times if a project is to be completed within its timescale. They became known as bar charts. Years later, the Dupont Chemical Co refined Gantt's ideas into critical path analysis (CPA). In the 1950's the US Navy developed a similar approach to CPA called project evaluation and program technique (PERT).

"Quite simply, software is an idea, and like all ideas it is intangible. Worse, its intangibility has not even been clearly defined.

CPA and PERT consist of network diagrams that represent the component activities of a project. The time needed for each activity and the dependencies between the activities are analysed. From this, the earliest and latest start and end dates for each activity can be specified.

Computerised versions of CPA and PERT are no strangers to the computer industry and software development. They have been in use since the early 1960s. But they have been most widely used in jobs other than software development. The fact is that the intangible nature of large software projects has, in the past, made them less amenable to the project management techniques widely used in other industry areas. The generally intangible nature of programs has made software project management especially complex.

One of the main problems in applying the discipline of these project management techniques has been the difficulty in defining the project lifecycle. If the specification starts off with fuzzy edges and then undergoes changes during the course of the project, it is exceedingly difficult to define a firm lifecycle.

TO AK UN X*

E SI R T

UN ER TA D,

W 'VE DD D

5 % M RE

NS RU TI N

C DE.

To re te IX,** BM as ak n t e U IX pe at ng ys em nd ad it as er o u e b ad in ha fa il io li es fi st uc io co e.

AI op ra es n t e I M 6 50 ic oc mp te, an no on he BM er on l S st m/2 od l 8, an bo h c nf rm o t e e er in PO IX ta da d.

Th ne op ra in sy te of er fo r d ff re t i te fa es: he wo ta da ds (-sh ll & ou ne he l), a OS ty e i te fa e, a d a im le en - dr ve in er ac . In dd ti n y u'l di co er ha po ti g U IX pp ic ti ns o A X i si pl , to .

On he BM 15 , AI in lu es BM' Di tr bu ed er ic s p og am. hi al ow yo to cc ss ıl s o a 6 50 ys em ha is si g A X, a if he we e r si en o y ur oc l s st m.

Th re re al a m ll on oo re so s w y A X w ll ak yo r l fe as er.

T f nd ut or wr te o A ne ew an, BM ni ed in do Lt .,Fr ep st, on on 4 5 R. O ca l h r o 01- 78 39 .

*UN X I A R GI TE ED RA EM RK FA &T.** IX S A RA EM RK FI TE NA IO AL US NE S M CH NE CO OR TI N.

This is particularly true in software projects because it is not always clear when a software project is due to end. The requirement for enhancements and maintenance after the formal end of the project can add huge extra costs and should often be regarded as part of the original software specification. The last 20 years or so has, therefore, seen a search for specific software project management tools that can solve these unique problems.

The Birth of IPSE

The search has been conducted on several levels. The first level – and the most visible aspect of the lifecycle – was programming. Structured programming languages, like Pascal, C and Ada, have made the programming process more controlled and systematic. Yet studies have revealed that anything up to 70 or 80 per cent of software lifecycle costs occur in maintenance. That is the result of errors and inadequacies arising from deficiencies in earlier requirements capture, design and analysis phases. So a new level of project management was needed. It has been found in design tools such as graphical-based system specification languages, structured system design methodologies and analyst workbenches.

Running in parallel with this, there has been a growing interest in creating improvements to the underlying infrastructure that supports the full lifecycle. As is so often the case with large scale projects, a management breakthrough came in the defence sector. In the late 1970s the US Department of Defense defined an Ada Programming Support Environment (APSE). The scope of this work was later broadened from programming to full project requirements, and APSE was generalised into an Integrated Project Support Environment (IPSE). The importance of IPSE is that it is able to encompass software toolsets, which may themselves embrace different methodologies, and provide the means to manage an intangible and fast changing project.

With IPSE the tiger of software management has at least been placed in a cage with real bars, even though he is not completely tamed. What helps to make him more docile still is the use of the toolsets that give project managers the computer aided software engineering tools they need to define, implement and control plans successfully. Typically, these toolsets will give managers the opportunity to control better a wide range of tasks as shown in Figure 1. In short, the use of IPSE with such toolsets provides managers with benefits in three main areas:

* The estimation of project timescales, costs and staffing requirements

* The planning of work in detail and adapting plans to take account of actual progress and changing requirements

* The control of developments through comprehensive and timely progress monitoring.

These benefits are seen in the outcome of the project. First, the software is likely to be of better quality. Improved predictability, control and monitoring ensures that. Then there are substantial cost savings. Using such techniques should result in reductions of at least 10 per cent in timescales and effort used. There could be an even bigger pay-off as managers become more skilled in using the techniques and tools. Of course, using these techniques successfully means devoting the necessary time to planning. Planning must be done on a realistic basis – and the plans must be capable of being changed or adapted when inevitable changes occur.

"IPSE encompasses varied tools and methodologies providing the means to manage intangible and fast changing projects."

Most projects will be planned on a hierarchical basis where each level describes the project in more detail. The lowest levels are the smallest separately manageable units of work which can be grouped into tasks. The higher levels in the structure are concerned with management and reporting structures, such as allocating groups of tasks to individuals and teams. The underlying techniques for controlling relationships between activities are still based on CPA and PERT methodologies.

Overall, the planning should encompass all the aspects of the project – people management, project structures, project methodology, team organisation and reporting mechanisms. Figure 1 illustrates project management functions that must be performed in controlling software developments.

Automating Tools

Computerising this planning process has an important management effect. It makes the project manager take the initial planning more seriously. He will probably be under pressure from management to deliver parts of the system quickly. He needs to balance those pressures against the need to plan the project effectively from the outset. Providing the discipline of computerised project management helps him to do this. Yet the existence of computerised project management tools will not do the manager's job for him. Development of the initial detailed plan is, in fact, the activity least affected by computerised tools.

The quality of management thinking and the painstaking human effort involved in defining project structures and activity networks, play a key role determining the outcome of the project. Nor will use of the computerised techniques work well unless managers have received training in their use. Managers need to appreciate and use the general principles of systematic project planning. And there are dangers in the tools themselves. Using incompatible tools for different phases may lead to considerable time spent formatting and transferring data between tools.

The primary objective of using such tools is to produce better software at lower cost. But there are plenty of other worthwhile benefits to be reaped. Although they will be of special use in large projects they can benefit small projects too. Using a consistent project management approach throughout an organisation can help coordinate smaller projects that have to be integrated into a whole. The approach will also assist managers and staff to move from one project to another with greater effectiveness.

Good project management can make life easier and more comfortable for software developers by giving them a clearer definition of how their performance will be judged. The computerised tools need not impose inhuman working practices – they can cope with flexible hours, for example.

Even the best computerised software management tools will not ensure that a project turns out successfully and on time. But they can at least give managers a head start. With software projects becoming increasingly complex, managers are going to need all the help they can get.

Anna Neale is with GEC Software and can be contacted on 01 345 6523.

TO MAKE UNIX* EASIER TO UNDERSTAND, WE'VE ADDED 50% MORE INSTRUCTION CODE.

To create AIX,** IBM has taken the UNIX operating system and made it easier to use by adding half a million lines of instruction code.

AIX operates on the IBM 6150 Microcomputer, and now on the IBM Personal System/2™ Model 80, and both conform to the emerging POSIX standard.

The new operating system offers four different interfaces: the two standards (C-shell & Bourne shell), a DOS style interface, and a simple menu-driven interface. In addition you'll discover that porting UNIX applications to AIX is simple, too.

On the IBM 6150, AIX includes IBM's Distributed Services program. This allows you to access files on a 6150 system that is using AIX, as if they were resident on your local system.

There are half a million good reasons why AIX will make your life easier.

To find out more write to Anne Newman, IBM United Kingdom Ltd., Freepost, London W4 5BR. Or call her on 01-578 4399.

IBM

*UNIX IS A REGISTERED TRADEMARK OF AT&T. **AIX IS A TRADEMARK OF INTERNATIONAL BUSINESS MACHINES CORPORATION.

CIRCLE NO 785

G and Auto-G – A Systems Notation with CASE Support

Goran Hemdal looks at a graphical language for real-time systems requirements specification.

G is a British notation language for formalising the often wooly requirements for software systems. Used properly, it can provide a complete, distortion-free model of real-time systems structure and behaviour. It is accompanied by a product called Auto-G which allows computer automation of the notation and support functions. G has a unique approach which led it to be the only non-US product to be considered for the SDI programme, including for use in developing an architecture for the battle management and command control part of SDI. This article explains the rationale behind G and how it works.

More Emphasis on Design

The development of a computer application involves more than producing the required program code. Programming languages and program design techniques, nevertheless, have been the focus of system development methodologies for many years.

Long experience with the general unreliability and high cost of building software, particularly for more complex systems, has created a growing awareness that programming advances are insufficient on their own. Improving the pre-programming phases of system development and optimising the overall engineering lifecycle are now becoming the centre of attention.

If programs are written to meet ambiguous, incomplete and inconsistent design specifications, an excessive burden is placed on the maintenance and enhancement end of the lifecycle. Such system developments have been the norm in the past. They lack good design visibility – a clear structure that runs through all design specifications and program code. Changes to the system are usually made by delving directly into the innards of code. The implications of making a change to one part of the code cannot be fully tracked, often giving rise to further errors and inconsistencies. The result of this situation has been that 70 to 80% of the total costs of system development have usually occurred in maintenance and enhancements activities.

Formal Notations and SDL

To overcome this, a firmer foundation must be established to system building. Its backbone comes from having a formal notation that can be used to develop unambiguous and complete specifications to give good design visibility. A new notation alone would be insufficient unless it also had effective support from Computer Aided System Engineering (CASE) tools, such as analyst workbenches.

For example, a design-orientated notation should have a graphical capability because specifications can be most lucidly expressed through functional diagrams. A CASE tool that can automate the laborious tasks of creating and maintaining such diagrams is essential to make use of the notation cost-effective. CASE tools improve the quality of design by providing computerised automatic checks on the correct use of the notation's syntax and on the logical consistency of design. The G language and Auto-G, its associated workbench tool is the solution proffered by a company called Advanced System Architecture.

G and Auto-G provide a formal and fully automated means of capturing user requirements in a rigorous way even when these may start from relatively vague conceptions of what will finally be needed. They allow the analysis of requirements to develop specifications of the functions to be performed and results expected, highlighting any ambiguities and incomplete descriptions inherent in the requirements definition. They further allow specification of the overall system design in terms of the structure, behaviour and communications that must be provided by the computerised implementation.

The idea of such a notation is not new of course. Various methodologies and notations have been used in the past for these requirements capturing, analysis and overall system design activities. They range from naturally imprecise languages, like English, to more systematic approaches, such as specification languages and structured design techniques.

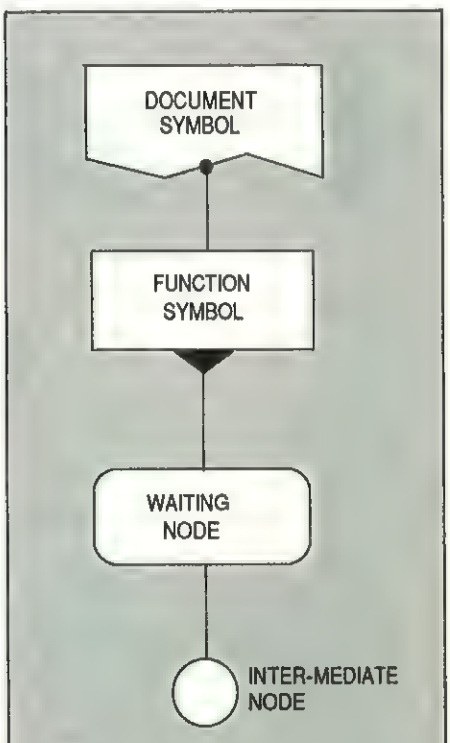


Figure 1: Overview of the main G symbols.

Figure 2: G Symbols in a design.

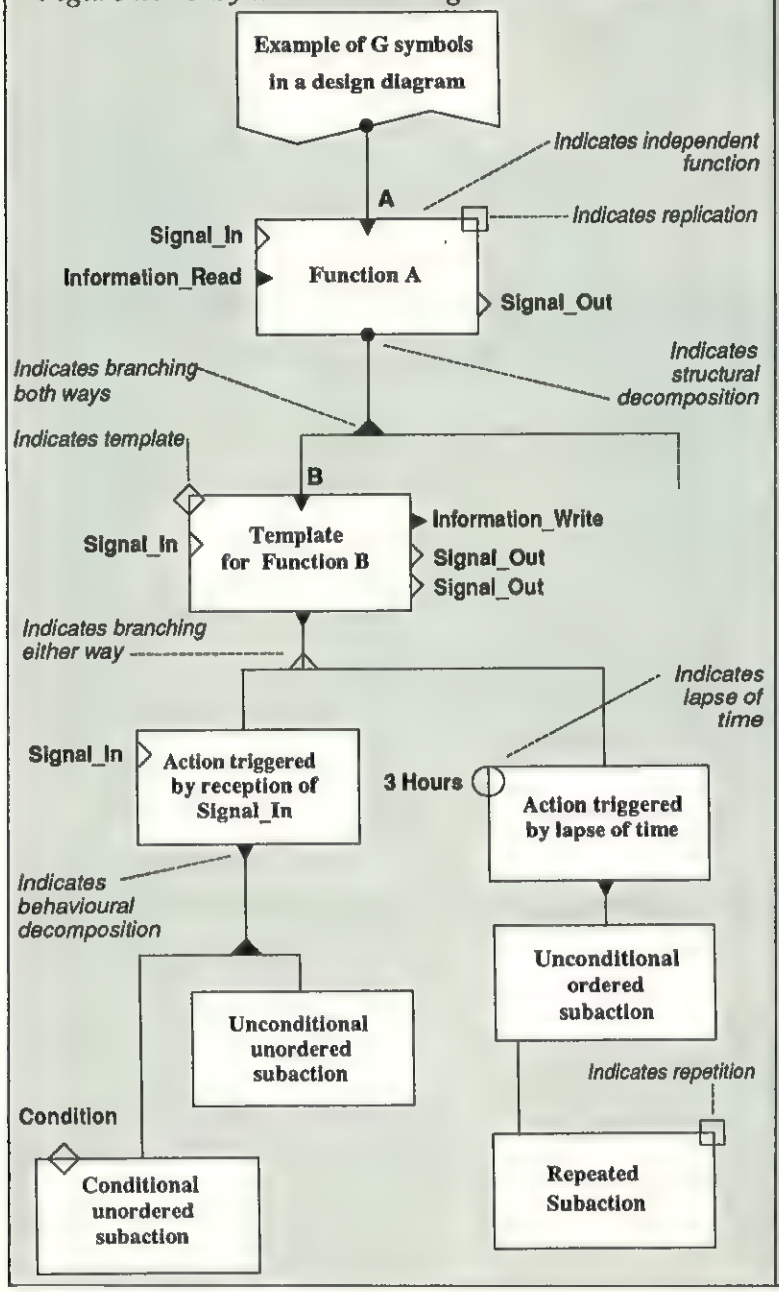
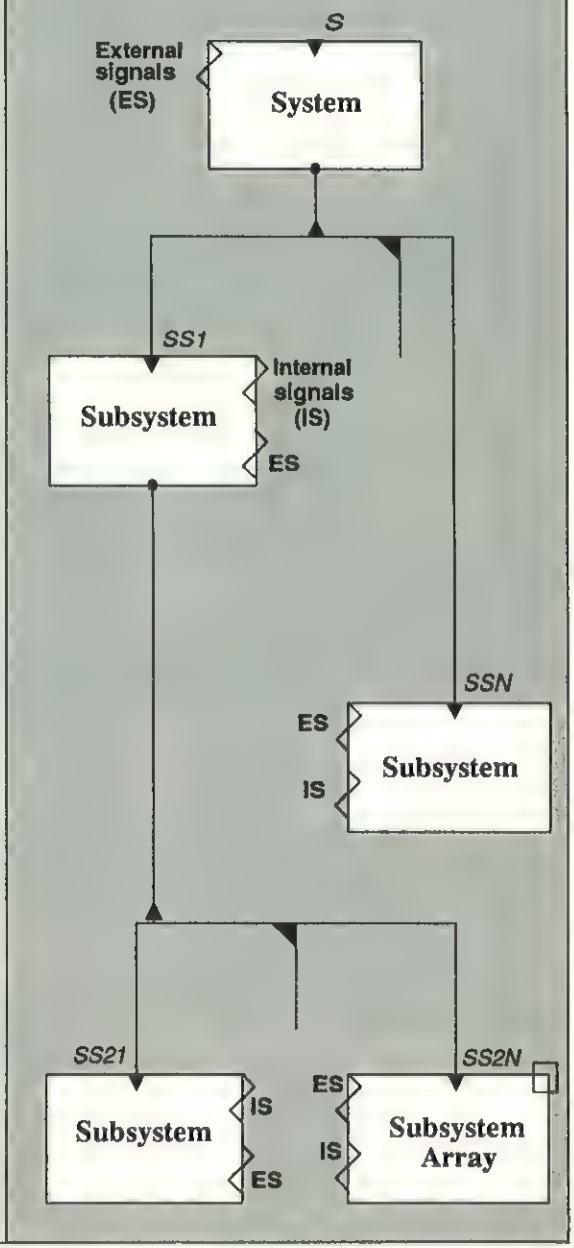


Figure 3: Model of a loosely-coupled system expressed in G.



No single method has provided a sufficient degree of formal rigour and ability to handle all relevant design concepts. Inconsistencies between notations and underlying methodologies used in different activities causes transitional errors between development phases. It also leads to serious distortions from real-world needs.

Some basic requirements for a specification and design notation were set out by the CCITT, the international telecommunications committee, when it started developing SDL (System Description Language) in 1972. SDL aimed to be:

1. Easy to use and interpret in relation to the needs of the organisation applying it.

2. Able to provide unambiguous specifications and descriptions.

3. Extensible to cover new developments.

4. Able to support several system specification and design methodologies, without assuming any of these in it basic capabilities.

These objectives are reasonable but are insufficient in themselves. They are difficult to assess because they depend on subjective judgements to determine how well they have been met. In addition to these SDL aims, G therefore also seeks to achieve the following fuller and more objective set of goals:

5. Permits distortion-free models of the real world to be represented. Any method that permits or imposes distorted models will be less easy to learn, use or interpret.

6. Allows complete functional representations of any system. This, together with distortion-free modelling of real-world needs, is needed to support different development methodologies. Notations that cannot express all necessary concepts can lead to incorrect design decisions being taken before the consequences have been adequately explored.

7. Provides processing by automated means (Auto-G).

8. Permits direct generation of target code of complete systems. Auto-G can, for example, generate code in Ada or C directly from G-based specifications. This helps to enable all system maintenance to take place at the highest design level, which is easier, quicker, more reliable and much less costly than making such alterations directly to program code.

9. Graphically based but with an equivalent textural form, T, which may be needed in some circumstances. It has few main symbols but many subsymbols can be attached to these to enable a rich range of system concepts to be expressed precisely.

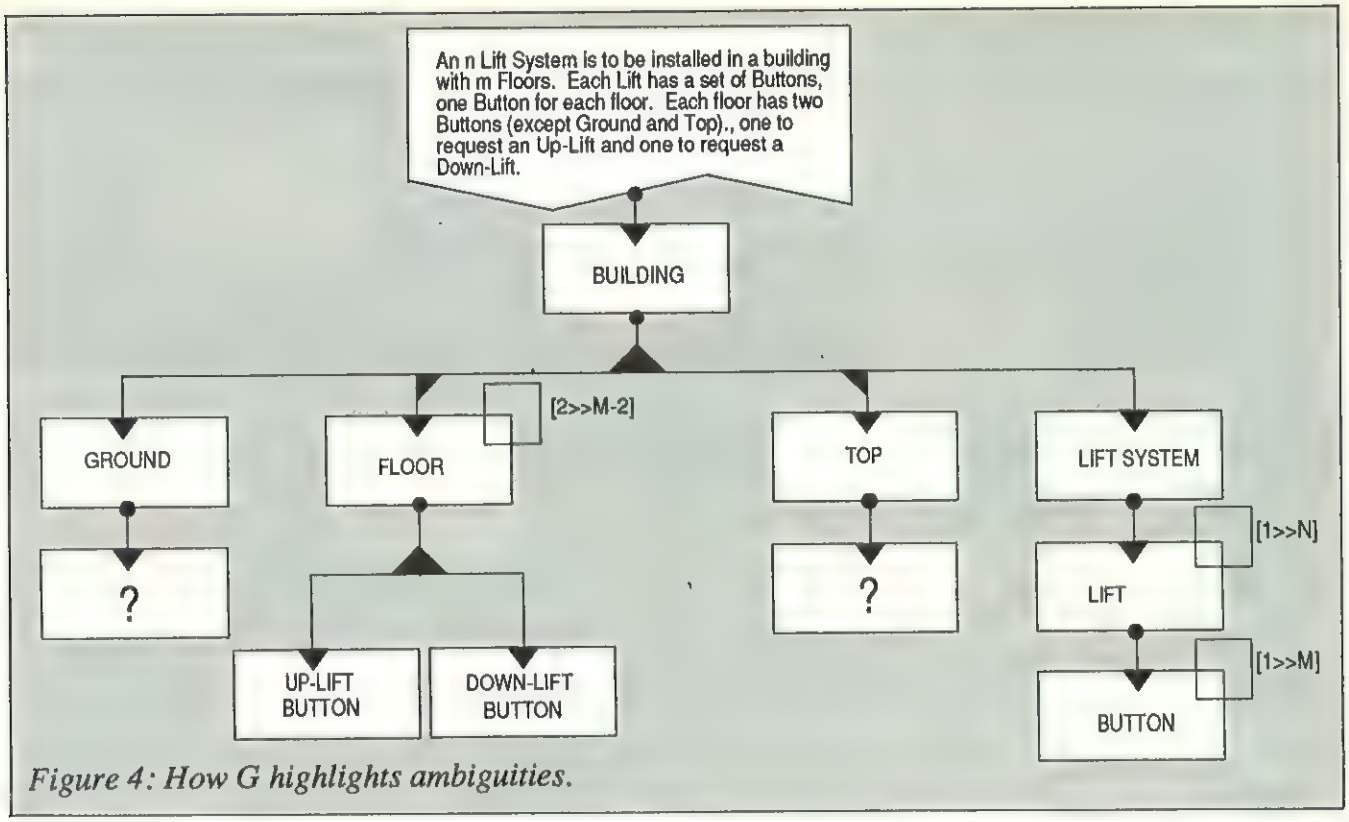


Figure 4: How G highlights ambiguities.

Figure 1 shows the four types of main symbol. A document specifies a named G module which consists of a design expressed in G. This symbol is also used as the off-page connector for linking parts of a large G diagram. The rectangle specifies the function performed by a system, subsystem, or a particular action or activity. A waiting node is used at a point where the system is suspended for a specified reason.

Figure 2 illustrates how some often used subsymbols can be combined in a G diagram. Those attached to the vertical sides of a function symbol indicate a variety of interface types, or triggering conditions, such as input and output signals, information reads and writes, and time lapses (which indicate an action is to be carried out after a given period of time). Symbols at the top and corners of functional boxes indicate the nature of the function, such as the fact that it is repeated a number of times or is a template, which could be equivalent to a program procedure.

As can be seen in Figure 2, there are also other ways of representing design characteristics. The 'function dependency' subsymbol at the top of Function A, for example, indicates that Function A is an independent element. Junction subsymbols indicate the branching or merging of relation lines, indicating, say, branching both ways or either way.

Applications

G can be used for any type of system but it has special advantages in real-time applications where the system must be always available to give immediate responses and analyses. Using ASA's strict functional decomposition discipline known as DEMOS (Decomposition Method for Object-oriented Systems), G can provide a complete, distortion-free model of real-time systems structure and behaviour.

A real-time system consists essentially of a number of autonomous functional units that operate asynchronously, ie with unexpected or unpredicted timing, rather than following a specific synchronous order. Functional autonomy means the system is loosely coupled, rather than having functions tightly locked together, say by sharing data.

Figure 3 illustrates the basic asynchronous loosely-coupled system model specified in G using the discipline or DEMOS and the automated aid of Auto-G. Each level contains successively more detailed descriptions of the systems. A design can be decomposed (broken down) until it reaches the smallest independent asynchronously operating entities, known as processes, which can only perform one action at a time.

Systems and subsystems, as illustrated in G Figure 3, interact with each other and the world outside via asynchronous events and reactions conveyed via signals carrying information about a single event. Such a system may simultaneously receive an arbitrary number of signals and carry out a corresponding number of actions as would be expected in a real-time system, such as aircraft controls or a financial trading telecommunications network.

DEMOS flushes out design flaws as soon as the designer starts building G specifications. Figure 4 illustrates how this occurs for a lift control system. The initial user requirement has been captured in narrative form in a G document. When these requirements are translated into G, ambiguities in the requirement are identified (shown by function boxes with question marks). It may seem obvious that Ground and Top floors should each have only one button ('Up' for Ground and 'Down' for Top). The rigorous DEMOS methodology insists that such requirement omissions are made visible so they can be resolved at the requirements capture stage. This relatively simple example illustrates the power of DEMOS and G. Resolving design problems at an early stage, without altering program code directly, can in itself dramatically cut software lifecycle costs by reducing maintenance needs. It also increases the reliability and adaptability of the system.

Auto-G

Auto-G turns the theoretical strengths of G into a cost-effective means of building computer-based applications. It is an analyst workbench that can be used with a wide variety of graphical workstations. Currently available Auto-G versions range from relatively low cost systems with relatively limited graphical capabilities, like the Atari 1040ST personal computer, to high performance devices, like Sun workstations. Auto-G can be run ona standalone workstation or accessed as part of a network or VAX-based cluster. Output can be made to low-cost graph plotters if necessary.

Auto-G ensures high quality G designs are implemented quickly. Quality is maintained because the G notation is machine checkable and Auto-G automatically implements various consistency and control requirements. For example, Auto-G eliminates the possibility of making syntax errors at the design stage because it prevents symbols and subsymbols from being used in an incorrect way. It also verifies the logical consistency of designs automatically. In large systems, one of the most difficult tasks is to control the various versions of a design that may be 'live' at any time. Documents, systems and subsystems go through many versions during development. Different teams may be working on advanced versions of the parts of the system they are developing, while other teams are sill referring to earlier versions. The necessary configuration and version control facilities needed to manage this problem successfully are an intrinsic part of Auto-G capabilities.

Auto-G's powerful Editor plays a role in helping to cut the time to create a system design by about 50%. It has what is widely accepted as essential features of an easy to use but powerful user interface. It is menu-driven, with context-sensitive pop-up menus presenting the user with the option to use only those symbols that are meaningful within the current context. Actions are initiated through mouse/icon interaction with multiple windows available if necessary. The Editor set includes a full range of functions, such as Change, Move, Delete, Hide, Explode, Reposition, Scale Up or Scale Down. Predefined layout rules can be permanently altered, for example by using a Scale command to adjust the relative size of items. The user's view can be temporarily manipulated. For example, some functions not currently of interest can be 'hidden' or the user can 'zoom' through the design to check its structure.

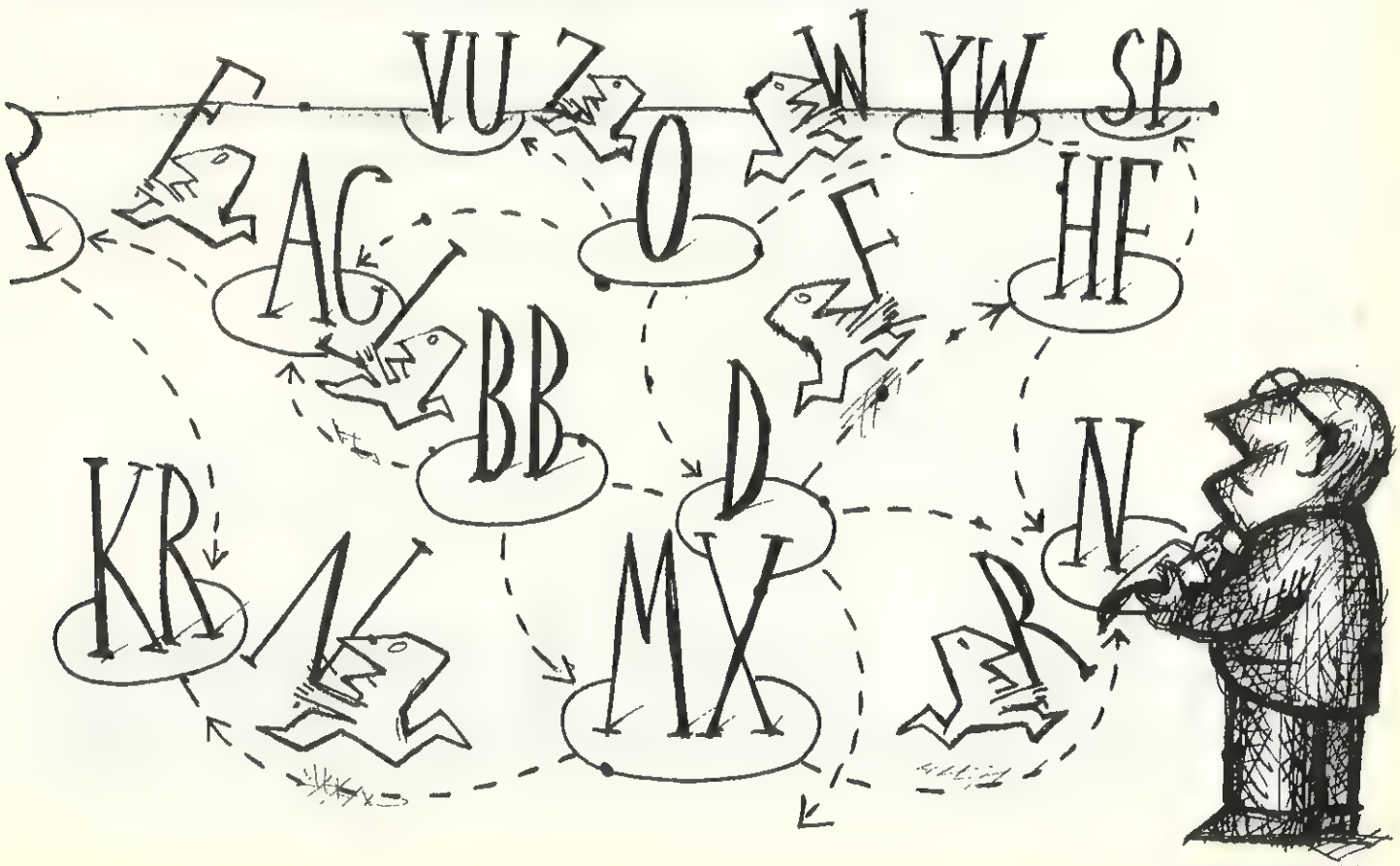
Auto-G helps to overcome two of the most error-prone, time- consuming and people-intensive phases of system development-:program coding and documentation. As already mentioned, Auto-G can produce code in target languages, like Ada and C, directly from G specifications. Accurate documentation can be produced automatically either in graphical (G) or textual (T) form. Auto-G can implement formal notational support for all requirements specification and design methodologies. These include MASCOT (Modular Approach to Software Construction Operation and Test), which is widely used for UK Ministry of Defence projects, and various Design Flow and Structured Analysis techniques.

Auto-G is designed to provide maximum flexibility in the types of output that can be produced from G designs. As well as direct production of code, it can also translate designs into other notations. And it is the first system that can automatically generate designs in the US Department of Defense's Strategic Defense Initiative (SDI) Ada Process Description Language (SA/PDL). This is a formal textural system design notation that must be used by contractors to certain SDI projects.

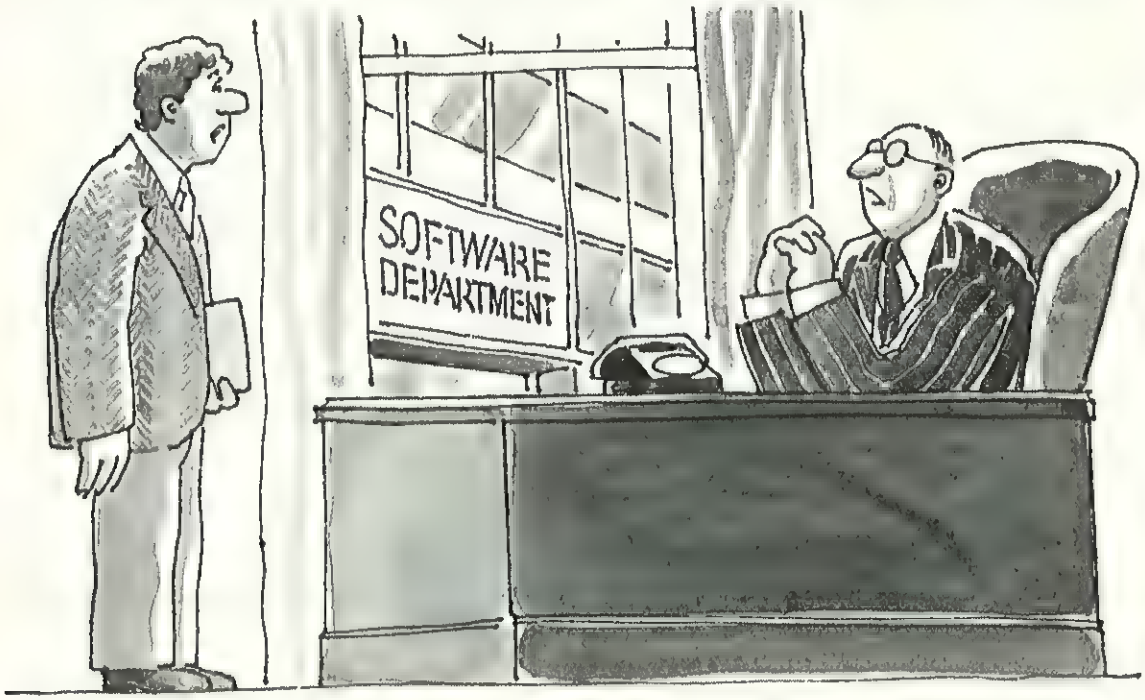
G and Auto-G are particularly in demand for ultra high reliability real-time applications requiring loosely-coupled architectures. Systems engineering aids and hardware support for these systems have been particularly poor in the past. Yet recent advances in hardware microelectronics and telecommunications has dramatically enhanced the scope and functionality of such systems. The G methodology supported by Auto-G has been used on a variety of real-time projects. These include air traffic control systems, the design of military vehicles, advanced telecommunications applications, real-time data sensors that can provide rapid fusion of large volumes of data from multiple inputs, the specification of integrated circuits, flight control for aircraft, space-based digital switching and sea-based radar.

Auto-G's unique approach led it to be the only non-US product to be considered for the SDI programme, including for use in developing an architecture for the battle management and command control part of SDI.

Goran Hamdal is the technical director of Advanced Systems Architectures (ASA) Ltd. He can be contacted at 0990 27111.



Development – the unmanageable in search of the impossible



"Another million pounds, George – certainly"

A lack of software development methodology can make projects difficult to control whilst lack of feedback means that management may have little true information of the real progress of a development.

GENOS™, GEC Software's Integrated Project Support Environment (IPSE) applies software tools to automate the whole software development process, ensuring that the day-to-day progress of software development can be accurately controlled and charted.

Clear, concise and understandable management reporting is just one of the many benefits that *GENOS*™ brings to complex software developments.

To find out how *GENOS*™ will increase management control and help keep costs to budgeted levels, raise productivity, improve communications and give better product quality, telephone 01-240 7171 or write to Alison Gould, GEC Software, 132-135 Long Acre, London WC2E 9AH.

If it's not *GENOS*™ – it's not an IPSE

(available for VAX/VMS & UNIX workstations)

gec software

A GEC Company

132-135 LONG ACRE, LONDON WC2E 9AH · TELEPHONE 01-240 7171 · TELEX 21403 GECLAG · FAX 01-240 9329

CIRCLE NO 786

Working Real-Time Extensions to UNIX

Peter Bond looks at the content and rationale of new extensions to UNIX to make the operating system suitable for real-time use.

There has been much debate on the role of real-time UNIX, but industry now seems to be converging around the need to formally establish a recognised standard. In the meantime, some proprietary real-time UNIXes exist. This article looks at one of these, from Ferranti Computer Systems.

The Ferranti-Originated Real-Time UNIX (Fortunix) was developed as part of the ESPRIT-funded DELTA-4 project. The prototype Fortunix has been working since February 1987, and is described in detail in the DELTA-4 specification 'Real-Time Extensions to UNIX', obtainable from Ferranti or other members of the DELTA-4 consortium. Fortunix will be released as marketable product as soon as certain enhancements have been added, principally to speed up the File Handling system. A brief summary of the main features of this product is provided below. A more detailed Functional Design Specification will be produced later.

Convergence with the evolving Real-Time UNIX standard is a recognised objective of the Fortunix development. Other objectives include the continued improvement of performance and dependability metrics.

Scheduling

A real-time process is one which interacts with real external activity and respects deadlines imposed by that activity. Fortunix provides mechanisms to enable real-time processes to achieve their deadlines. These deadlines may vary from microseconds to hours. If a real-time process has a required response time measured in microseconds, it must run as an interrupt service routine. Millisecond response times, however, can be achieved by memory-resident user processes running under Fortunix.

Two real-time processes may differ in the required probability that their deadlines be achieved: they are then given different priorities. A Fortunix system may also contain processes without deadlines: these also may be given different priorities, depending on their different throughput constraints. Fortunix processes are therefore given a static priority and, optionally, a specified deadline. They are scheduled in the order of their priorities, but a group of processes with the same priority may be scheduled in two different ways:

1. If they have specified deadlines. This gives the highest probability that all the deadlines will be met.

2. If they have no specified deadlines, they are given equal timeshares by a simplified form of standard UNIX timesharing. This is appropriate for processes subject to constraints such as fair timesharing or minimum throughput over a period.

The same principles are used in the scheduling of other system resources such as memory (except for real-time processes which have locked themselves in memory). Real-time processes waiting for the same semaphore, or waiting to lock the same file, will also succeed in the same priority/deadline order, and non-real-time processes in a FIFO order, not the LIFO order of the UNIX System V. Users are not required to predict the future execution times or activation times of their processes. However, if such prediction is possible, they can ensure that their processes will be scheduled at the planned times by using system calls described in the next section.

To help processes achieve their deadlines, Fortunix schedules them pre-emptively, i.e. the current process is made to suspend as soon as possible after a more urgent process becomes ready to displace it. Standard System V UNIX does not schedule pre-emptively, and can cause delays of the order of a second before the more urgent process runs. Fortunix has so far reduced this pre-emption delay to a maximum of 2 milliseconds: the goal is 1 millisecond.

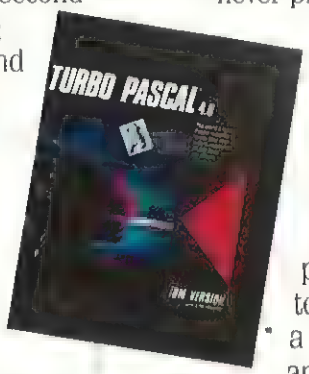
The interrupt latency, or delay, before an interrupt service routine is entered, has also been reduced. The scheduler overhead, or time for the scheduler to select a process, has been kept to a minimum by the simplicity of the scheduling algorithm and of the data structures used. The context switching operation, when the current state of a process is saved or restored, is implemented in assembler code and is probably as fast as possible.

If there is insufficient processing power, processes will sometimes miss their deadlines, and periodic processes may still be running at the start of the next period. Fortunix then sends them a missed deadline signal, which the process may ignore, may be killed by, or may catch and process. For example, a process running every 10 msecs may learn in this way it has taken more than 10 msecs to complete its periodic processing. It might react by raising its priority or increasing its period to 20 msecs.

A real-time service may be implemented by several separate real-time processes, running sequentially, concurrently or in parallel. Fortunix ensures that all these processes inherit the priority and deadline assigned to the real-time service. However, a process may also change its priority and deadline provided its owner is an authorised Real-Time user. (see below)

Turbo Pascal 4·0 – faster an

Our new Turbo Pascal 4.0* is so fast it's almost reckless. How fast? Better than 27,000 lines of code per minute†. That's more than twice as fast as Turbo Pascal 3.0.
And Turbo Pascal 4.0's user friendly, intelligent design provides a second to none Pascal programming environment for beginners and professional alike.



4.0 Technical Highlights:

- Compiles 27,000 lines per minute
- Includes automatic project Make
- Supports > 64K programs
- Uses units for separate compilation
- Integrated development environment
- Interactive error detection/location
- Includes a command line version of the compiler
- Highly compatible with 3.0

*For the IBM PS/2 and the IBM and Compaq families of personal computers and all 100% compatibles.

4.0 breaks the code barrier

No more swapping code in and out to beat the 64K code barrier. Designed for large programs. Turbo Pascal 4.0 lets you use every byte of memory in your computer.

4.0 uses logical units for separate compilation

Pascal 4.0 lets you break up the code gang into "units", or "chunks". These logical modules can be worked with swiftly and separately. 4.0 also includes an automatic project Make.

4.0's cursor automatically lands on any trouble spot

4.0's interactive error detection and location means that the cursor automatically lands where the error is. While you're compiling or running a program, you get an error message and the cursor flags the error's location for you.

You'll get everything you need from Turbo Pascal 4.0, its Tutor and its 5 toolboxes.
In fact, the Turbo Pascal family is all you'll ever need to perfect programming in Pascal. If you've never programmed in Pascal, you'll probably want to start with Turbo Pascal Tutor.
With TURBO TUTOR 4.0, you'll learn Pascal from the people who invented TURBO PASCAL. Turbo Tutor steps you from basic right through advanced programming concepts and techniques. The package includes a 400-page, quick-study tutorial and the accompanying disc contains source code for every complete example in the manual.
Turbo Tutor gives a concise history of Pascal, defines basic terminology and helps the person new to programming start writing Turbo Pascal programs. As your expertise quickly gains, add Toolboxes like our.

- Database Toolbox
- Editor Toolbox
- Graphix Toolbox
- GameWorks
- Numerical Methods Toolbox

If you develop Turbo Pascal programs for end-users, you can distribute your own compiled programs that include all or part of the above toolboxes — with no royalty payments.

Turbo Database Toolbox

Now you don't have to "reinvent the wheel" each time you write new Turbo Pascal database programs.
TURBO DATABASE TOOLBOX 4.0 enhances programming with Turbo Access and Turbo Sort.

Sieve (25 iterations)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Size of Executable Files	2224 bytes	11682 bytes
Executive speed	9.3 seconds	9.7 seconds

Sieve of Eratosthenes, run on an 8MHz IBM AT

Since the source line above is too small to indicate a difference in compilation speed we compiled our CHESS program from Turbo Gameworks to give you a true sense of how much faster 4.0 really is!

Compilation of CHESS PAS (5469 lines)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Compilation speed	12.1 seconds	Turbo Pascal 3.0
Lines per minute	27,119	9,243

CHESS PAS compiled on an 8MHz IBM AT

†Run on an 8MHz IBM AT

d brighter than anything you've known

TURBO ACCESS quickly locates, inserts or deletes records in a database, using B + trees — the fastest method for finding and retrieving database information. Source code is on disc.
TURBO SORT, using the Quicksort method, sorts data on single items on multiple keys. Turbo Sort features virtual memory management for sorting large data files. Available now with commented source code on disc.
Your TURBO DATABASE TOOLBOX comes with source code for a free sample database — right on your disc the database can be searched by keywords or numbers. Update, add or delete records as needed. Just compile it, and it's ready to go to work for you. You can tailor it to your specific needs.

Turbo Editor Toolbox 4.0

All you need to build your own text editor or word processor.
Turbo Editor Toolbox gives you easy-to-install modules. Now you can integrate a fast and powerful text editor into your programs. You get the source code, the manual and the know-how.
We provide all the editing routines. You plug in the features you want. You could build a WordStar-like editor with pull down menus like Microsoft's Word, and make it work as fast as WordPerfect.

Turbo Graphix Toolbox 4.0

A library of graphics routines which is so simple to use it even lets complete beginners create high-resolution graphics on the IBM PC, true compatibles, and the Zenith Z-100. It gives you a set of tools to include in your programs for complex business graphics, easy windowing and storing screen images to memory.

Turbo Gameworks 4.0

Explores the world of state-of-the-art computer games. Using easy-to-understand examples, Turbo GameWorks teaches you techniques to quickly create your own computer games using Turbo Pascal. Or for instant excitement, play the three great computer games we've included on disc — compiled and and ready to run.
Chess, Bridge, Go-Moku. We give you these three classic games of strategy. And we give you the source code so you can see how computer games are designed.
Turbo GameWorks provides state-of-the-art games that let you be player, referee, and rules committee — because you have the Turbo Pascal source code.

Turbo Pascal Numerical Methods Toolbox 4.0

Add Numerical Analysis to your Turbo Pascal Development Systems. The Turbo Pascal Numerical Methods Toolbox is an indispensable tool for all scientists and engineers. As a complete collection of Turbo Pascal routines and programs, the Toolbox provides you with all the state-of-the-art applied mathematics tools you'll ever need — so you don't have to reinvent the wheel every time you write a program.
The Turbo Pascal Numerical Methods Toolbox does for Turbo Pascal users what the IMSL and NAG mathematical routines do for mainframe FORTRAN users.
The Numerical Methods Toolbox offers ten powerful features:

- Zeros of a function
- Interpolation
- Differentiation
- Integration
- Matrix Inversion
- Matrix Eigenvalues
- Differential Equations
- Least Squares
- Fourier Transforms
- Graphics

Again, source code is provided for all programs.

For the dealer nearest to you, or to order now.
Call 0734 320022

BORLAND INTERNATIONAL

I want to benefit from Turbo Pascal 4.0 and the 4.0 Toolboxes.

If you are a registered Turbo Pascal user and have not been notified of version 4.0 by mail, please call us at 0734 320022. To upgrade if you have not registered your product, just send the original registration form from your manual on payment with this complete coupon to:

Borland International UK Ltd, 8 Pavilions, Ruscombe Business Park, Twyford, Berkshire RG10 9NN

Copies	Products	Price	Total
☐	Turbo Pascal 4.0	£89.95	
☐	Developer's Library (includes Tutor and all Toolboxes)	£245.00	
☐	Turbo Pascal Tutor	£49.95	
☐	TNT (Turbo Pascal 4.0 and Tutor)	£129.95	
☐	Database Toolbox	£69.95	
☐	Graphix Toolbox	£69.95	
☐	Editor Toolbox	£69.95	
☐	Numerical Methods Toolbox	£69.95	
☐	Turbo Gameworks	£69.95	
☐	Turbo C	£89.95	
☐	Turbo Basic	£69.95	
☐	Turbo Prolog	£69.95	
upgrades:			
☐	Turbo Pascal 4.0	£30.00	
☐	Developer's Library and Pascal 4.0, (For Jumbo Pascal owners only)	£150.00	
☐	Turbo Pascal Tutor	£20.00	
☐	TNT (Turbo Pascal 4.0 and Tutor)	£40.00	
☐	Database Toolbox	£20.00	
☐	Graphix Toolbox	£20.00	
☐	Editor Toolbox	£20.00	
☐	Numerical Methods Toolbox	£20.00	
☐	Turbo Gameworks	£20.00	

Add 15% VAT
Amount enclosed _____
Prices include shipping to all UK cities.

My computer's name and model is:

The disc size I use is ☐5¼" ☐3½"
Payment: Access/Visa/Money order/Cheque
Credit Card expiration date: _____
Card no

Name _____
Signature: _____
Shipping Address: _____
Name _____
Postal Code: _____
Telephone: _____

CODs and purchase orders WILL NOT BE accepted by Borland

All prices are suggested list prices and are subject to change without notice.

NOT COPY PROTECTED

CIRCLE NO 787

Extensions to UNIX

The Fortunix extensions are all additions to standard UNIX facilities. UNIX System V remains a subset of the system, and can be used to develop, maintain and analyse the 'Real-time Subsystem', as well as to absorb spare processing capacity. All users have access to this UNIX V subsystem, but the real-time subsystem is accessible only to authorised real-time users who acquire their status by being admitted to the real-time users' group by the UNIX super-user. These protective arrangements are an addition to the standard UNIX protection of files and programs based on different access codes for different classes of user. A real-time user may create real-time processes and give them a real-time priority and optional deadline (see above). Real-time users and processes may also issue a number of real-time commands and system calls, which are briefly summarised below.

1. They may lock a real-time process in memory until it terminates, and/or keep it in swap space even after it has terminated. On UNIX V, these privileges are available only to the super-user.

2. They may read the current system time, or the remaining time till the deadline or start the next period. All the time intervals are measured in units of 1 millisecond on the Fortunix prototype, but the time unit may be reduced if higher resolution is required. A process may also read its current priority.

3. A process may specify a periodic reactivation time in milliseconds. Unless the system clock interrupts every millisecond, the specified period will be rounded up to a multiple of the clock interrupt period, which may be any integral number of milliseconds. At the start of each specified period, the process will be unsuspended if it is pausing, or sent the missed deadline signal if it is still running. A shell command, for example, may be executed at a time specified to the nearest second. This is an improvement on the 1-minute resolution of the UNIX System V.

4. A process may suspend itself until a specified time or for a specified period in milliseconds. It will be unsuspended at the next clock interrupt after the specified time. (This system call is now also available to non-real-time processes).

5. Deadlines are also specified in the usual time units (milliseconds on the prototype) and rounded to the next clock interrupt time.

6. A process may send a UNIX message after a delay specified in milliseconds.

7. A combined **fork and exec** system call has been added, which creates a new process with a specified priority without terminating the calling process, effectively combining the UNIX V system calls **fork**, **exec** and **nice**. This saves overheads such as the unnecessary copying of process data to a child process which only does **exec**.

8. System calls have been added for fast instantiation of a process which is already waiting in memory or swap space in a suspended but immediately executable state.

9. System calls are being added to suspend or unsuspend a child process, and change its priority. The parent process can then manage 'frame scheduling' of a number of child processes required to start and stop at specified times in a 'time frame'.

10. System calls are being added to speed up file handling and support a subset of AT&T's proposed General Events Mechanism (see below).

Inter-Process Cooperation

Inter-process communication, synchronisation and mutual exclusion are more important in a typical real-time application than in the time-sharing applications in which UNIX has been most often used. We need IPC mechanisms which are efficient, deterministic (not sensitive to slight changes of timing) and distributable (since real-time systems make increasing use of parallel processing to achieve their deadlines).

Standard UNIX V has many inter-process cooperation features, notably pipes, signals, semaphores, messages and shared memory, but none of these are easily generalised to a distributed environment. For example, in their standard form all of them have identifiers which are unique only within a single UNIX system, not on a network of such systems. (See inset on BSD 'Sockets' in 'RPC: The Key To Distributed Software' in this issue). Files however can be generalised to such networks, because they can be given such network-wide identifiers. FIFO files can therefore operate as network-wide pipes.

UNIX signals suffer from some non-deterministic features, which become more noticeable and potentially dangerous in a real-time application. For example, a process can set itself up to catch a signal and then pause, assuming the signal will awaken it from the pause. After this has worked correctly many times, it may catch the signal *before* entering the pause, and then it pauses for ever. If a process sets itself up to catch a signal, and is sent two signals in rapid succession, the second signal may either be lost altogether (since there is no count of signals) or may be handled in the default fashion because the process has not yet reset itself to catch the second signal as well.

AT&T have proposed a General Events Mechanism, to overcome these limitations. This uses event queues with names in the file name space, so that they can be created , opened, protected and distributed like files. They contain events queued in FIFO or priority order, which may be received either like signals or like messages. It is possible to wait for a conjunction or disjunction of events. Events may contain any amount of data, which may be passed in shared memory.

Fortunix therefore retains standard UNIX IPC, but also includes a subset of the AT&T General Events Mechanism, which is so comprehensive that it can support every IPC function from distributed semaphores to deterministic signals, all accessed via a uniform user interface.

Asynchronous I/O

Using the General Events Mechanism, a real-time process can initiate I/O and then turn to some other processing, receiving an event when the I/O is complete. Indeed AT&T also propose a system call interface for asynchronous I/O which may become the real-time UNIX standard.

UNIX already does asynchronous disc writes via cache buffers. However, some users require an assurance that the file they have just written is actually on the disc, without losing all the extra throughput derived from buffering. Fortunix therefore offers a new system call which ensures that a specified file has been written to disc.

Fast File Handling

Fortunix continues to support two standard UNIX file interfaces:

1. The standard SVID interface for users and applications, to ensure portability of application software.

2. The layout of exchangeable discs, so as to maintain a data interchange capability with other UNIX systems.

However, the standard disc layout arrangements optimise usage of disc space rather than speed of file access. This is not suitable to real-time applications, whose ability to meet their deadlines may depend critically on the speed of access to their files. Access times also need to be deterministic, which suggests

SOFTWARE DEVELOPMENT TOOLS

PVCS

The Most Powerful & Flexible Source Code Revision & Version Control System

The Polytron Version Control System (**PVCS**) allows programmers, project managers, librarians and system administrators to effectively control the poliferation of revisions and versions of source code in software systems and products. **PVCS** is a superb tool for programmers and programming teams. (A special LAN version is also available.) If you allow simultaneous changes to a module **PVCS** can merge the changes into a single new revision. If the changes conflict, the user is notified. Powerful capabilities include: Stores and retrieves multiple revisions of text; Maintains a complete history of revisions to act as an "audit trail" to monitor the evolution of a software system; Maintains seperate lines of development or "branching"; Provides for levels of security to assure system integrity; Uses an intelligent "difference detection" to minimize the amount of disk space required to store a new version. Requires DOS 2.0 or higher. Compatiable with IBM PC,XT,AT and other MS-DOS PC's.

Personal PVCS
For single-programmer projects **£99.00**
Corporate PVCS
For larger, multiple-programmer projects **£249.00**
Network PVCS from **£665.00**

UNIX Tools on DOS

MKS Toolkit

Over 110 programs that perform tasks on machines like the IBM PC,XT, or AT with the ease that one would expect while working under UNIX. Designed especially for those developing software in a DOS enviroment, these utilities include:

vi - UNIX full screen editor
awk - data transformation & report generation language
prof - give a profile of the execution times of a command
egrep - find a sting using full regular expression patterns
diff - find the difference between two files

cat	**chmod**	**cp**	**comm**	**cut**	**date**	**dd**	**dev**	**df**
du	**echo**	**ed**	**file**	**find**	**head**	**help**	**join**	**lc**
line	**ls**	**more**	**mv**	**nm**	**od**	**paste**	**pwd**	**rm**
sed	**sh**	**size**	**sort**	**split**	**strings**	**tail**	**time**	**touch**
tr	**uniq**	**wc**						

and more ...

The programs come with a shell and complete UNIX-style command line file name expansion on 3 DSDD 5.25" diskettes, load and run under DOS. and are not copy-protected. Full documentation is included.

MKS Toolkit version 2.2
Only £ 99.00 excluding V.A.T.
Call for four page fact sheet

dBASE to C
the dBx Translator

- dBx produces quality C direct from dBASE II,III or III+ or Clipper programs.
- Move dBASE programs to UNIX or other machines.
- Improve program speed and reliability.
- Support multi-user/network applications.
- Includes full screen handler and allows a choice of C database managers.
- May be used to move programs, help dBASE programmers learn C easily, or as a daily tool.
- For MSDOS, PCDOS, UNIX, XENIX, Macintosh, AMIGA

PRICES EXCLUDING V.A.T.
dBx dBASE to C translator **£250.00**
dBx Portable version includes library source **£495.00**
dBx with library and translator source **£995.00**
CALL FOR dBx FACT SHEET

CATALOGUES

- Scientific and Engineering Software
- dBASE™ Development Tools and Utilities
- Software Development Tools
- Public Domain C Source Code

Telephone or write for your copy of one of the above catalogues and to be included on the mailing list for our new monthly newsletter **TOOLBOX™**

A COMPLETE DATA ACCESS SYSTEM
C-INDEX+

C-INDEX+ is not just a B-Tree, or an ISAM, or a file manager, **C-INDEX+** is an advanced system that provides a complete set of tools for storing and retrieving any kind of data. **C-INDEX+** has a variety of access views, from the highest level of fully automated multi-keyed record storage, to the actual low-level building blocks the rest of the system is built upon. **C-INDEX+** also has a list of features that make it unmatched in the industry. With this combination of features and flexibility, **C-INDEX+** eliminates the need for any other kind of file access.

FEATURES

- Variable length keys and data
- Compile and run-time definition
- Automated multi-key storage
- Complete with full source code
- True multi-user
- Keys and data in one file
- Written in K&R C
- No royalties

ALL THIS FOR ONLY £175.00

CALL FOR FACT SHEET AND DETAILS OF OUR ONE DAY DEVELOPING DATABASE APPLICATIONS IN C WORKSHOPS USING C-INDEX+ TO BE GIVEN BY ITS AUTHOR CHRIS DEPPE.

THE SOFTWARE CONSTRUCTION COMPANY LTD.
28 CHURCHMEAD, ROYDON, HARLOW, ESSEX CM19 5EA
(027 979) 2566

CIRCLE NO 788

the file should be memory- resident in the most critical cases.

Fortunix therefore offers an alternative file storage system, optimising speed of file access rather than usage of memory and disc space, at least for real-time users and their files, though it could in principle be made available to non-real-time users. The principal features of this file storage system are described below.

1. **A larger disc cache** is used than on a standard UNIX V, and a greater divergence is tolerated between disc pages and more up-to-date cache pages. In addition to the standard 'sync' command to flush the whole cache onto disc, a more selective 'flush this file only' is provided.

The cache should ideally be stored in a stable or battery-backed memory so that its contents survive a power failure. The UNIX kernel start-up routines are modified to recognise a cache which survived a power failure, and flush its contents to disc before re-initialising it. If some cache buffers occupy unstable memory, they should only be used to hold read-only file data, not updated data or control information.

The UNIX **plock** system call, used to lock a processes code or data in memory, is extended so that it can also lock its file data in the cache. A particularly critical real-time process may use this to ensure a file is memory-resident for as long as necessary.

2. **Two allocation block sizes.** Space on both the disc and the cache is allocated in large blocks known as 'chunks', which are 4K-16K bytes, and in smaller blocks called 'fragments', which are 512-1024K bytes, as is on UNIX System V. Files created by real-time users and processes are stored in the more efficient chunks, while directories, indices and lower priority files are stored in fragments so that they waste the minimum of disc and cache space.

For real-time files, the 'read ahead' and 'write behind' speed-up effects of the cache are thus increased, and the number of disc transfers during sequential access is greatly reduced. Access to large files also benefits because they no longer require a 3-level tree of index blocks, but can now be mapped by a single index of chunk pointers until they are very large (more than 2 Mbytes for 8K chunks).

The System Administrator targets the priority level above which processes create files which are stored in chunks rather than fragments. However, when the disc is nearly full, warning messages are generated and low priority processes will start to be denied further disc space. After this, there is no guarantee as to which allocation block size the system will use for a file, the greater the probability that chunks will be used rather than fragments.

3. **Pre-allocation of disc space.** Exceptionally, a non-real-time file should also be stored in chunks, or a real-time file may be so small that it can be stored in a single fragment. A user may indicate these exceptional cases by estimating the likely size of the file immediately after creating it. The system then chooses a suitable allocation block size and also pre-allocates disc space up to the estimated file size, which speeds up subsequent write accesses.

4. **Optional implementation in a parallel processor.** The hardware proposed for the first release of Fortunix includes a separate close-coupled processor to manage all disc/cache-resident files.

5. **Privileged treatment of high-priority processes.**
Cache space, like all other system resources, is allocated preferentially to higher priority processes, and when two processes have the same priority, to the process with the earlier deadline. So data being accessed by high priority processes remains in the cache for longer than the file data of low priority processes. To be certain of finding its file data in the cache, however, a critical process should lock it in memory using the extended **plock** system call.

As both disc and cache fill up, the remaining free space is reserved for higher and higher priorities. When the disc is nearly full, the system therefore issues a series of warning messages: 'Disc full for priority N processes', undergoing a sort of 'graceful degradation'.

A system flushing process flushes cache buffers to disc, doing so at a priority which rises as the cache fills. Buffers containing closed files are flushed first, then buffers being used by processes with priority up to that of the system flushing process. The highest priority, most recently used buffers will be flushed last, but the precise algorithm is still the subject of detailed design and experiment. Sometimes a process will demand cache space when only free cache space is reserved for higher priority processes. In this case the system flushing process must flush buffers of lower priority than the calling process.

Disc transfers are ordered on the priority of the calling process. When choosing between disc transfers of the same priority, the disc driver may execute the transfers in FIFO order, or optimise throughput using the geometry of the disc.

A process which has locked a file has its priority temporarily raised to the priority of any process waiting to lock the file. This reduces the interlock delay suffered by high priority processes.

6. **A low-level 'numbered file' interface.** Fortunix provides an interface for transient files which are never named, to save the overhead directory searches. The files are identified by their INODE number, and the user may access their cache buffers directly, removing the need to copy between user space and the cache. This interface is suitable to kernel processes requiring very fast access, such as the memory management system. It is not suitable to portable applications since it is not standard. On the first release of Fortunix, virtual memory management remains independent of file handling.

7. **Duplication of the Super-block.** It is particularly important to protect the UNIX super-block during power failure. Fortunix therefore either duplicates it in the second sector of the filesystem, or in a battery-backed cache.

Loadable Device Drivers

Real-time application programmers are frequently required to write or modify device drivers for new peripherals or telemetry equipment. They may sometimes want to write these device drivers as real-time user processes and load them into memory only when they are needed, without regenerating the UNIX kernel. Such a 'loadable' device driver normally contains an interrupt servicing routine which has to be linked dynamically to the hardware interrupt vector when the device driver is loaded. Device addresses also have to be linked dynamically into the address space of the program. This facility is a considerable extension of the traditional UNIX device driver linked statically into the UNIX kernel, and may not be provided as part of the first release of Fortunix. A Manual on 'Writing UNIX Device Drivers', aimed at Real-time programmers with limited experience, will also be produced.

Peter Bond is with Ferranti Computer Systems in Manchester, and may be contacted on 061 499 3355. References in this article are 'A Proposal for a General Events Mechanism' by S J Buroff and C F Schimmel, published by AT&T Information Systems (1987).

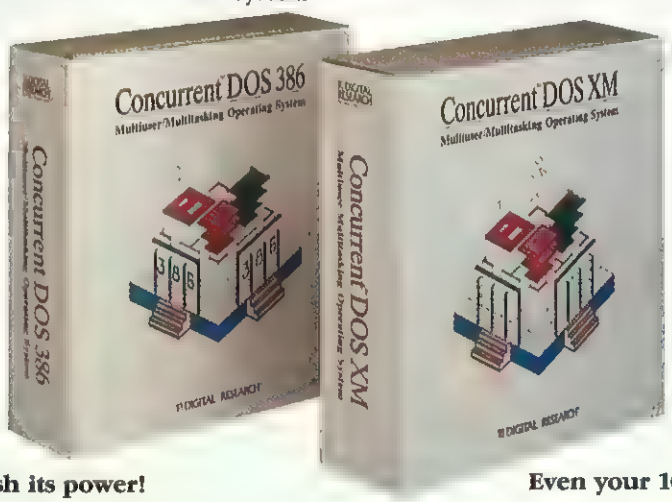
Turn your PC into a Powerstation

With the multiuser multitasking Concurrent™ DOS family

Concurrent DOS 386

Allows you to harness the full potential of the Intel 80386 microprocessor. As a 386 PC user. As an OEM. As a system builder. Concurrent DOS 386 uses the 80386 architecture to allow up to four gigabytes of address space. Provides compatibility with popular PC DOS single user and Concurrent multiuser applications. The standard retail pack turns your 386 machine into a central powerstation distributing high performance support to 3 users while multiuser packs let you configure for up to 10 users.

Choose the 386 and unleash its power!



Concurrent DOS XM

Gives extended capability to users working with Intel 8086/80286 based machines. It allows you to run popular PC DOS programs simultaneously as well as a wide choice of multiuser Concurrent software including accounting, database, word processing or project management. You can take full advantage of expanded memory cards to work in up to 8 Megabytes of address space. Standard retail packs provide 3 user support while 6 user configurations are available via multiuser packs, from our Authorised Concurrent Dealers.

Even your 16-bit PC can be a Powerstation.

The Concurrent DOS Family

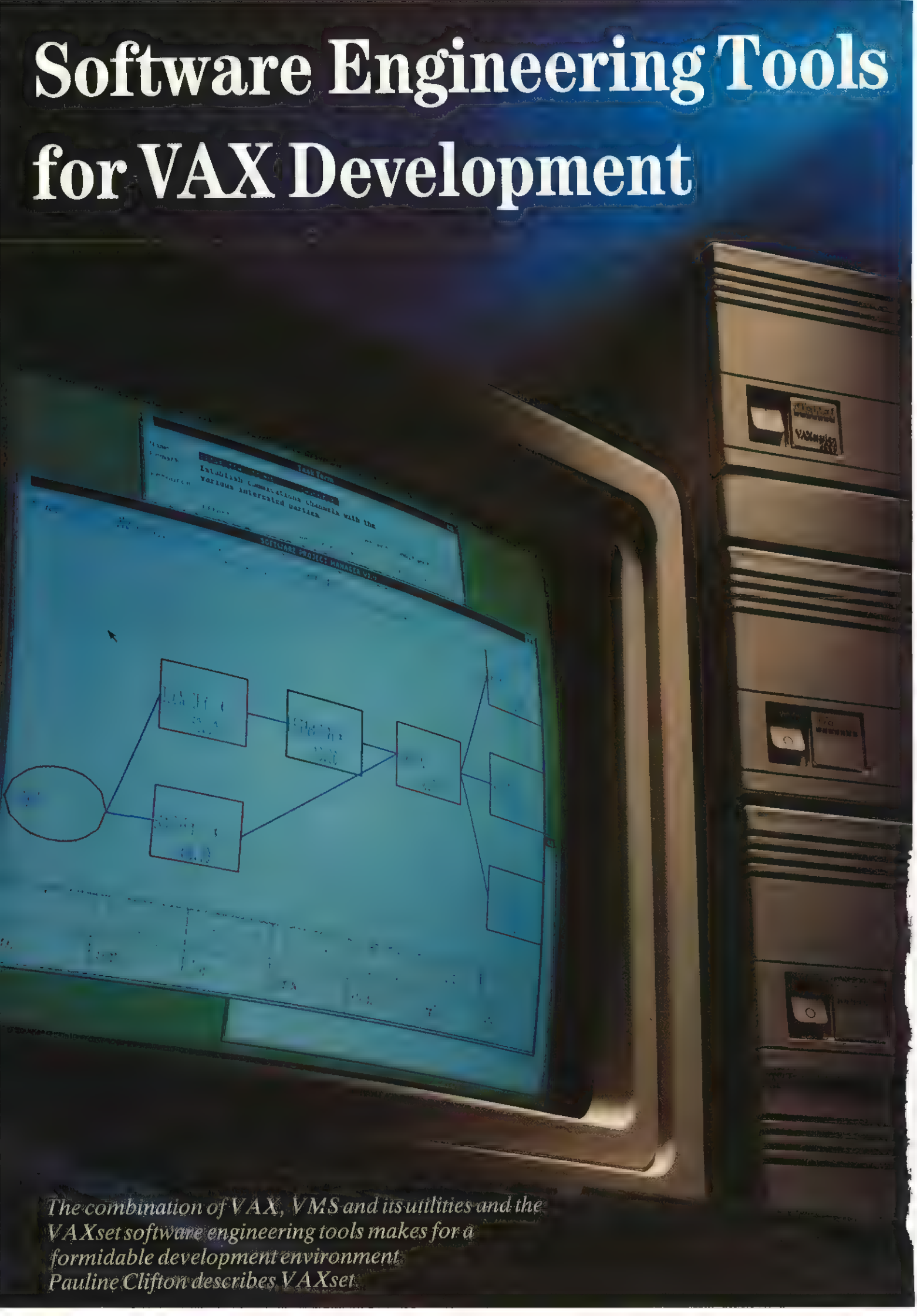
- Provides full multiuser multitasking.
- Runs existing PC DOS applications.
- Easy to manage system operation. No system manager needed.
- Toolkits available for ISVs, OEMs and VARs for system building and programming.
- Runs the same single or multiuser software on 8086, 80286 or 80386 PCs without change.

Concurrent DOS derives from Digital Research's decade of experience developing multitasking, multiuser operating systems for microprocessor-based designs. Telephone (0635) 35304 for your information pack.

DIGITAL RESEARCH

CIRCLE NO 789

Software Engineering Tools for VAX Development



The combination of VAX, VMS and its utilities and the VAXset software engineering tools makes for a formidable development environment. Pauline Clifton describes VAXset.

To many programmers, the ideal software development life cycle begins with a 'conception' phase where mental pictures take shape and ideas evolve. This then moves on to a testing, where ideas are coded and run to see if they work. Then there will be an integration phase where various code sections are pieced together to make a working monolithic program. Finally, there will be debugging. Lots of it, where the program is run repeatedly to check for problems until it works.

And a jolly good life cycle it is too. Small microcomputer applications will most often be written in this way and the closeness of the conception phase to the coding phase helps create a life and excitement which make the finished programs interesting and innovative. Large DP systems may well be written in this way too, provided that the programmers are divided into small teams with freedom to conceive and implement their own ideas.

But there are two situations which make this life cycle horrendously unworkable: when a project gets **large** or **critical**. A large project, maybe spanning five or ten years, needs a mechanism to unify all members of the team – the specs, deadlines, coding practices and so on must all be consistent. It makes a project manageable as work on many different programs which are ultimately part of the same total is carried out over many years and by different people. In this case, the freedom of the individual programmer is less important that the coherence of the group, and the programmer must learn to live with the 'that's the way it's got to be done' school of management.

A critical project is where performance times and predictability are of paramount important. A payroll package will not be as critical as a flight control system, for example. In fact, by comparison, payroll systems are not critical at all. The life cycle collapses here as code interfaces, variable naming conventions, performance and a heap of other factors contrive to make the finished software unreliable in severely taxing conditions.

For these large and/or critical projects, the life cycle should look something like this:

1 Requirements Specification (Analysis)

2 Software Design

3 Coding

4 Integration and Test

5 Documentation

6 Maintenance

Notice how 'requirements specification' (equivalent to the process of conception above) is formalised in a way that makes it complete and workable. Software design takes these ideas and puts them in a form which specifies the software to be written. Coding is the act of turning the design into runnable software. Integration is ensuring that the system works to specification on the target hardware. This does not mean shipping copies out to customers and asking for beta testing to be done. The potential cost of 'beta testing' flight control software systems, for example, might be beyond most developers' budgets! No, it means matching finished software models with specification and finding correlation between the two. It means testing automatically, each line of code with simulated inputs and outputs, and a whole lot more besides. Finally, notice how documentation and maintenance appear on the list!

*There are two situations where the conventional life cycle breaks down: when projects get **large** or **critical**."*

The DEC VAX is a popular development workhorse in these larger, critical projects and DEC offers a number of software tools to encourage and automate parts of the development cycle. The VAXset range comprises six products which are useful components in their own right and work well together as an integrated team. The VAX Performance and Coverage Analyser, for example, is a comprehensive code tester and the Language Sensitive Editor is a powerful editing tool. Other tools in the range are the VAX Source Code Analyser, a Code Management System, Module Management System and a Test Manager.

As individual aids to productivity, these six products are discussed below. Perhaps their biggest contribution, however, is in the way information from any one can be used by the other modules in the suite, creating a cohesive systems for the management and control of entire projects in a VAX environment. Most VAX languages are supported, and full access is given to the standard VAX development tools such as RMS, the Common Run Time Library and the VMS Debugger. The products look and feel similar, so that a programmer moving through the cycle will be able to pick up and use all modules relatively easily.

Code Management System

The Code Management System (CMS) tracks the creation, authorship and update history of any text file, giving a complete status report on all files in a project at any one time. Most of these files will be program modules, though the CMS provides the same tracking facilities for documentation files, plans and specifications also.

CMS is invoked by typing CMS at the VMS $ prompt. It responds by indicating what the current CMS library is and changing the prompt to **CMS**›. A file is created using: (See Figure 1).

The file is now referred to as an element, more specifically the first generation of that element. Subsequent changes to the file (usually from the LSE) will create second, third and so on generations.

CMS does not stop two people from updating (or 'reserving') the same file simultaneously, though the last in will be warned by CMS what is happening. If he insists in going ahead, he runs the risks of his changes conflicting with the other person's changes. The usurper's changes are saved not as a new generation file, but as a variant file. At this point, the CMS library will list generations of **SPEC.RNO** as: (See Figure 2).

CMS can incorporate John's changes into subsequent generations of the element while ensuring that these are consistent with those of Mike using: (See Figure 3)

If conflicts had occurred, these would have been flagged by CMS and the reserved element would have needed manual editing to resolve the conflict. The reserved file is saved back into the CMS library using: (See Figure 4) and the CMS library now looks like Figure 5

Access permissions permitting, anyone can list the most recent versions of files. Other CMS commands can list all elements in the system, elements of one file type, a transactional history of the entire system or of one element or either over a specified time period. A transactional history of certain kinds of operation can also be retrieved: (See Figure 6).

Language Sensitive Editor

This is a sophisticated editor/debugger masquerading as a rather ordinary text editor. Once loaded, the program can load in any source file, will recognise the language and immediately becomes sensitive to that language: it checks syntax and will recompile and reedit from within a single editing session. It also gives help on syntax for that language.

The LSE 'environment' file helps to implement a team's coding conventions or design standards by storing templates in binary form. These templates can be set up by the user to create a very customised editing environment, for example to store, for example, syntax variants, to support a new language. Equally, these templates can be used to specify the format of text-based reports, specifications and so on.

Elements from the templates are extracted using 'tokens' and 'placeholders', a shorthand notation. These can be redefined in situ, while editing. A simple **GOTO BUFFER/CREATE** command is issued, followed by **EXTRACT TOKEN**. The definition of the selected token is then edited and the **DO** command is issued to execute the new definition. For example, adding a **BEGIN/END** construct to the default definition of a Pascal **WHILE** statement, the following **EXTRACT** statement is issued:

```
LSE› EXTRACT TOKEN WHILE/
        LANGUAGE=PASCAL
```

The results of this are shown in Figure 1. **BEGIN** and **END** are added and the comment **WHILE** is added on the **END** statement. **CTRL/Z** returns to the LSE› prompt. Now, each time the **DO** command is issued, the new definition is executed and every time the **WHILE** token is used in the current Pascal editing session, LSE provides the new definition.

Source Code Analyser

The SCA gives a means of cross referencing the use of symbols, identifiers and references to identifiers. It also gives a means of visualising program layout: it can display relationships between routine calls, between symbols and between files. It can also analyse routines for consistency by checking what numbers and data types of arguments are passed and the types of arguments returned.

A developer new to a project could use SCA to learn the definitional use of a data structure, the declaration or call of a routine, coding standards and programming techniques. The following command could locate calls to the routine **BUILD-TREE**:

Figure 1: Extracting a token in the LSE

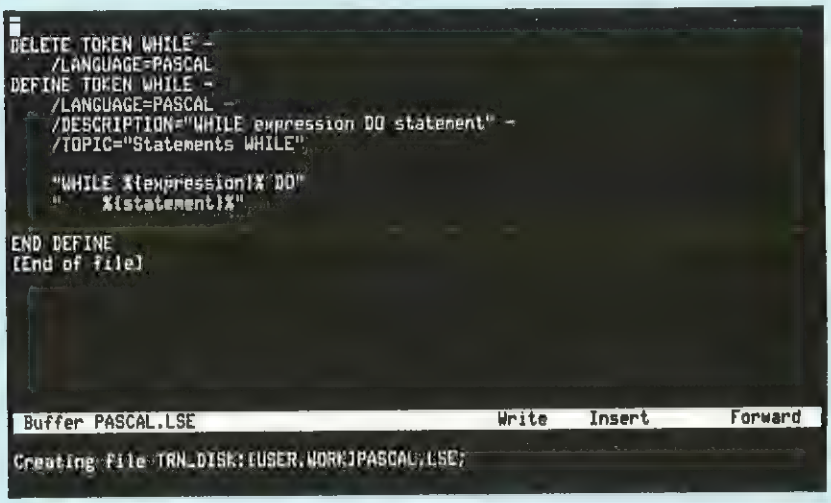
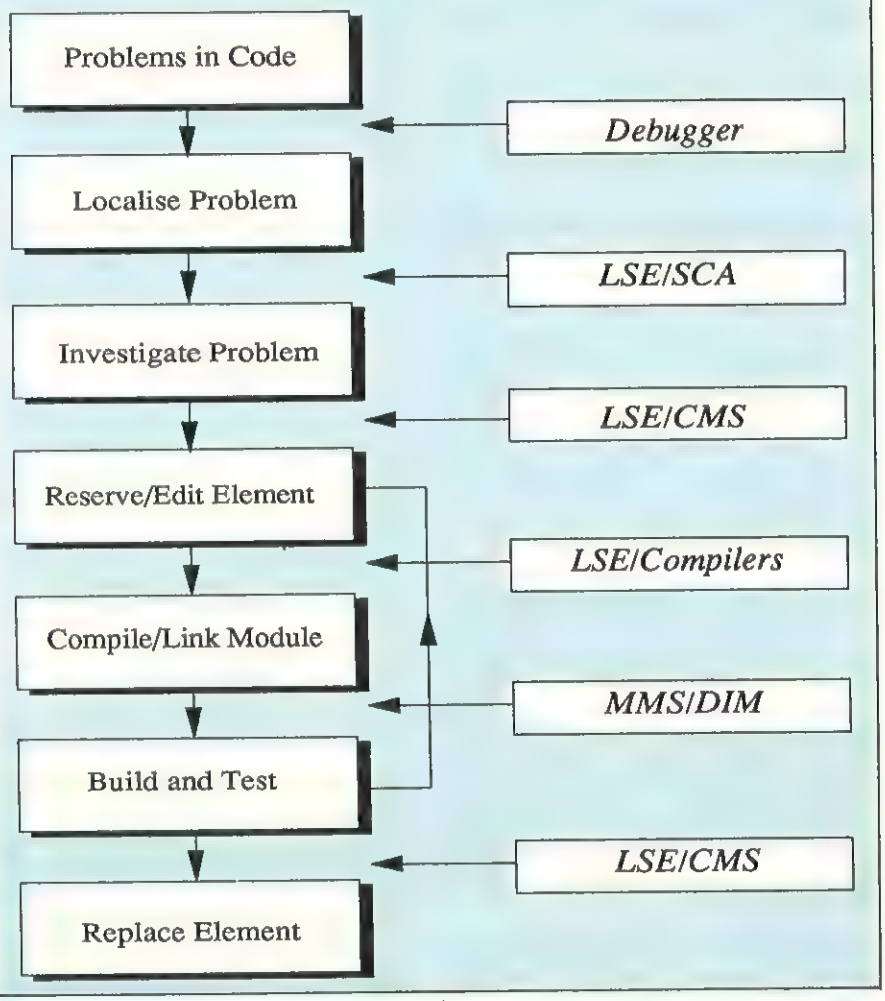


Figure 2: Building Code with SE Tools




```
LSE› FIND /REFERENCE*call
                build-tree
```

or display routine call information in Figure 7).

The Module Management System

This is a means of recompiling and linking source code files in an automated way. The files belonging to a particular system are specified in an MMS description file. This is an ASCII text file containing rules that describe how the components of the system are related. MMS then takes these components and builds the system. The time saving feature comes from the way in which it will recompile only those components which changed since the previous build. In this way, rebuilding a large system when only one small program file has changed is not an unnecessarily complex or lengthy procedure.

Test Manager

The VAX DEC/Test Manager (DTM) creates descriptions of tests carried out on software and can then execute specific tests or groups of tests. Results of tests are stored, and when that same test is carried out a second time, DTM checks that performance at least matches that obtained previously. This can help spot program errors and ensures that additions do not 'regress' performance. A variety of test and result reports can be produced from DTM.

The test description is the central element to a good test system. It consists of fields whose contents point to files needed to run the test. And the core of a test description is a template file. This is a DCL command procedure or a recorded DTM session file which exercises software. Each test has a template file.

Prologues and epilogues are command files associated with a specified test. The run before and after the template file. The prologue file begins by establishing any special environment.

The idea of grouping tests provides a further level of management. Test descriptions can be organised into groups which share a common characteristic or function. They could even be grouped under programmer name, or both name and function. If the system is rebuilt daily (or nightly) DTM could selectively run critical tests if they were grouped as such. Another group might contain tests which were important for quality assurance, thus allowing this subset to be run as required.

Performance/Coverage Analysis

The PCA analyse the run-time behaviour of software locating execution bottlenecks, finding code that is never executed by test data, showing what systems services are called, listing RMS I/O calls and other program activities. It also gives timing details, showing how much time is spent in any one procedure and can count the number of times a program executes specified locations.

Other Tools

Of course, the VAXset tools are an addition to the standard ingredients of VMS, such as the interactive symbolic debugger. The VAXset tools can work with the debugger so that, for example, the LSE can be called from within a debugging session to modify code for further debugging.

Table 2: Sample Debugging and Editing Session

A large project, the 'Transliteration Project', has moved into active coding. The team has already accumulated a body of code consisting of multiple module stored in a CMS library. This is supplemented with a read-only Reference Copy area. As an aid to analysing source, they have set up a project-wide SCA library; all developers also use their local SCA libraries as necessary for small tasks.

The utility under development, TRANSLIT, replaces all lowercase letters (a to z) with their uppercase counterparts (A to Z). The command looks like this:

```
TRANSLIT file-spec original-characters
                              [replacement-characters]
```

TRANSLIT has been updated and given a quick run-through test which showed no problems. The developer dumped is back into the CMS library. Full DTM testing later on, however, revealed the new code to be less than complete. Sometimes it works, sometimes not. A new programmer joins the team and is given the problem to solve. Equipped with the Debugger to step through the code, and some sample test data, the recovery begins.

The debugger allows flow of data from an input file to output file to be studied and this reveals that the first item of data exits the TRANSLIT utility, but others don't. Clearly, the program is not taking a proper branch the second time around. The **EXAMINE** command allows the examination of the destination line by line. Various debugging commands, such as **EVAL/DECIMAL** give a quick on-line aid the checking ASCII and decimal and soon the problem is traced to the variable TABLE[CODE].TRANS_VALUE having a value 258, which is an illegal ASCII character - this could cause the incorrect branching!

At this point, it is possible to enter the LSE with the command **EDIT/EXIT**, placing the source code on the screen at the same point highlighted in the debugger as shown below.

```
- SRC: module COPY_FILE -scroll-source-----------------------------------
   67:              THEN
   68:                  BEGIN
   69:                  out_index := out_index + 1;
   70:                  out_line[out_index] := CHR (table[code].trans_value);
   71:                  END
-> 72:              ELSE IF table[code].trans_value = newline
   73:              THEN
   74:                  BEGIN
   75:                  WRITELN (out_file, SUBSTR (out_line, 1, out_index));
   76:                  out_index := 0;
   77:                  END;
- OUT -output-------------------------------------------------------------
COPY_FILE\IN_LINE:      'aBcDeFgHiJk'
COPY_FILE\OUT_LINE[1:1]:        'A'
COPY_FILE\CODE: 66
66
COPY_FILE\TABLE[66]
    TRANS_VALUE:        258
    COMPRESS:   False
- PROMPT -error-program-prompt-------------------------------------------
DBG> EVAL/DECIMAL 'B'
DBG> EXAMINE table[code]
DBG> EDIT/EXIT
```

Table continued overleaf...

The link between the two tools allows the debugger to position the cursor on the line in the LSE source code corresponding to the line currently being debugged.

VMS Mail is a standard means of communicating between and within groups, and in programming teams is perhaps all the more useful.

The Common Data Dictionary (CDD) holds all data descriptions and definitions used by various VAX languages, including 4GLs, databases and tools.

Datatrieve, scan and Notes are three other tools that may be useful. Datatrieve is a query language to retrieve, store and modify data and can be used within applications programs. Scan is a string manipulation language to convert an input stream into a desired output stream. Notes provides a means of building a bulletin board type of system, of storing, indexing and retrieving notices left on the bulletin board. DEC recommends Notes be used by the development team as a means of supporting users and logging faults and problems.

Conclusion

The key element of the VAXset tools is their integratability. With the exception of the LSE, I would say that this was the prime strength of every individual tool. So VAXset meets the Project Management criteria of the development lifecycle. It also covers the areas of coding, documentation and maintenance well, and with the utilities of VMS, which include a database and data dictionary, the offering is certainly wide. The widespread use of these products in practice, however, is not something on which I can comment yet. I have not used all the products or talked with users, though I hope to return to this in a future issue of .EXE.

I would pick out the absence of tools at the 'requirements specification' end and 'integration' part of the spectrum, not as a flaw, but as a pointer to what should come next. There certainly is a gap here in DEC's offering. But then, that's one of DEC's strengths: what they don't supply, others will, and there are plenty of good cross compilers and SA/SD tools now on the market.

Information in this article was drawn from a DEC publication entitled 'A Methodology for Software Development Using VMS Tools' dated April 1987, with permission. Pauline Clifton is a researcher at the University of Reading.

This turns out to be the command on line 72 which clearly is giving a wrong evaluation. Having moved the cursor to the symbol, the keys CTRL D effectively issue the SCA **GOTO DECLARATION** command which shows how the symbol TRANS_VALUE is processed. Source shows TRANS_VALUE to be of type CODE_VALUE and so information on CODE_VALUE is gathered by positioning the cursor and hitting the CTRL D keys again. This displays its declaration and shows it to be defined between MIN_CODE and MAX_CODE. Using the same CTRL system, MIN_CODE is found to define UNDEF_CODE as 258 - the value found earlier in TABLE[CODE].TRANS_VALUE when using the debugger.

The SCA **FIND** command is then used to find all places where TRANS_VALUE is assigned a value and its results can be seen below.

```
        END;
    FOR code := min_code TO max_code DO
        BEGIN
        IF table[code].trans_value = undef_code
        THEN
            BEGIN
            table[code].trans_value := replace_code;
            table[code].compress := TRUE;
            END;
        END;
Buffer BUILDTABLE.PAS                    Read-only   Nomodify      Forward
  Symbol          Class        Module\Line          Type of Occurrence

TRANS_VALUE       component    BUILD_TABLE\76       write reference
                               BUILD_TABLE\100      write reference
                               BUILD_TABLE\107      write reference
                               BUILD_TABLE\131      write reference

Query 1  FIND TRANS_VALUE/REFERENCE=WRITE                      Forward
1 occurrence found (1 symbol, 1 name)
4 occurrences found (1 symbol, 1 name)
137 lines read from file TRN_DISK:[TRN.BLD_V1.WORK]BUILDTABLE.PAS;2
```

This highlights where a FOR loop is needed - a loop that was inadvertently removed by the previous programmer. This loop must set the values in TABLE that corresponds to characters that are not being translated.

To date, the source code has only been available in read-only form and a working copy must now be reserved with the **RESERVE** command. Working on this, tokens are used to generate the FOR, BEGIN and IF templates, and only loop-specific information need be entered. A sample token is:

```
FOR code := min_code TO max_code DO
    BEGIN
    IF %{expression}%
    THEN
        %{statement}%
    %[ ELSE %{statement}% ]%;
    %[statement-list]%...
    END;
```

The LSE **COMPILE/REVIEW** command then compiles the new code (without errors). The **MMS/CMS WORK_BUILD** command is issued to build the system, running the project's DTM test automatically. DTM is then used to review these tests. Fortunately, this time, they were correct.

Meet the Industry's New Standard for Mainframe C Compilers

SAS Institute Inc. announces a mainframe version of the Lattice® C Compiler—your key to truly portable applications.

With our compiler, you can develop C programs on IBM 370 machines, interface easily with non-C programs and software packages, and protect your programming investment across operating environments. Virtually every new computer supports C, and portable programs created with the mainframe compiler under OS or CMS will run on any other machine with a C compiler.

The mainframe compiler uses standard IBM linkage conventions. Assembler programs, MAIN routines in other high-level languages, and packages such as IBM's ISPF and GDDM can be invoked directly from C.

And you can use C, instead of assembler, to develop small and fast subroutines called from other languages.

We designed the compiler listing and cross-reference to make programs easy to follow and errors easy to find. An extensive library offers functions from Kernighan and Ritchie and the Lattice PC C compiler. The run-time library produces explicit numbered error messages and a traceback of active function calls if an error occurs.

For all the facts—including details on economical licensing complete with technical support and enhancements—call your Software Sales Consultant today.

SAS®

SAS Software Ltd.
Wittington House
Henley Road, Medmenham,
Marlow, SL7 2EB.
Tel.: 06284-6933
Telex: 846681
Fax: 06284-3203

UNIX-Hosted Real-Time Development

As one of the favoured development environments, UNIX is actually poorly suited for the development of real-time systems. Leslie Kirby, Jerry Fiddler and David Wilner describe a set of tools for BSD UNIX and 68000 to change this state of affairs.

Although using UNIX in a real-time environment has long been a dream of systems builders, that dream has remained elusive. The problem is the fact that UNIX was not designed to be a real-time system. In its 'standard' form, it lacks even the most rudimentary of what are normally considered to be real-time features. Yet, UNIX does have distinct advantages which make it a superb development system and a useful resource in a real-time environment.

Rather than try and combine the features of both systems into one, this article looks at software which allows a fully configured real-time system, complete with bus, controller cards and I/O to run with a fully configured UNIX system – either as a separate host, or as a board on the same system. The software is called VxWorks. Facilitating communication between the two systems is the key to VxWorks' approach. This article describes the UNIX/VxWorks development environment and how communication between the two is achieved in both networked and multiprocessing environments.

UNIX and Real-Time

Despite considerable efforts by some parties to make UNIX into a real-time system, it remains inappropriate for serious real-time use. It lacks many of the attributes that any real-time system should have, such as preemptive scheduling, the ability to lock processes in memory, a very fast task context switch and the ability to easily connect tasks to interrupts.

Although some of these features may be added, there are still further problems. UNIX isolates the applications from the hardware and from each other. While this may be appropriate for timesharing systems or workstations, real-time applications often need to be 'closer' to the hardware. Also, UNIX requires considerable hardware resources while many real-time situations call for one or more small (often diskless) systems. Finally, because UNIX is large and performs many functions, its response time is unpredictable. One can never know with certainty exactly how long any operation will take.

However, this is not to say that UNIX does not have a useful role to play in a real-time environment. .EXE readers will be all too familiar with the richness of the UNIX development environment and the abundance of programmers.

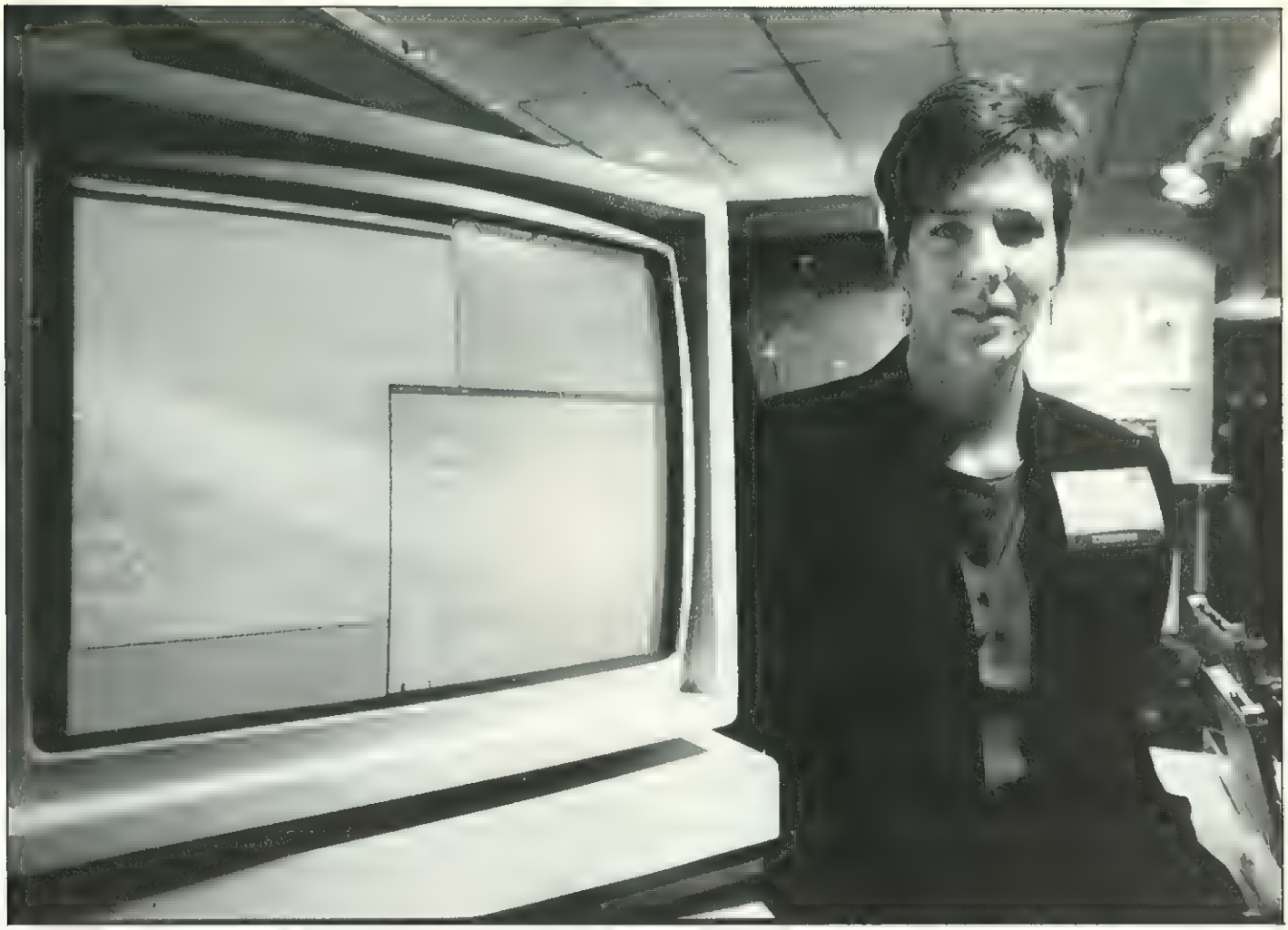
Looking more to the future, it's possible to see UNIX being used more as a component of a distributed real-time system to perform non-real-time functions, while controlling or communicating with other machines performing the actual real-time chores. Real-time systems often include such non-real-time components. For instance, a real-time computer controlling a robot might itself be controlled by a UNIX system running an expert system, or a number of real-time systems running a factory might be connected to UNIX systems tracking inventory or generating reports.

Real-time development and debugging present somewhat different problems than those which UNIX was originally designed to solve. In addition, real-time systems are becoming more and more complex, and thus more and more difficult to develop and debug. Which is where something like VxWorks comes into its own.

What Is VxWorks?

VxWorks is a true real-time operating system and support environment. It is a set of comprehensive routines which support a real-time kernel of the users choice – currently VRTX and PSOS are supported kernels. The routines allow for communica-

Figure 1: VxWorks Components.



Kirby: "UNIX is a vital distributed real-time system component."

Table 1: How VxWorks Supports Multiprocessing.

One of the issues in real-time distributed systems has been multiprocessing – the ability to work with multiple tightly coupled CPU's. Although hardware, in the form of powerful single-board computers, has become affordable and readily available, multiprocessing software has lagged behind.

VxWorks provides multiprocessing as a simple extension to a network. By providing a network driver that uses a backplane, such as a VMEbus, as the physical medium, VxWorks is able to provide all of the powerful communications tools already discussed to communicate between multiple CPU's in a single chassis. Thus, two CPU's sharing a backplane can communicate using sockets, **rlogin** between each other, etc. In addition, because of IP's transparent internetwork facilities, a CPU can act as a gateway between another network, such as Ethernet, and other CPU's sharing the backplane with it. This allows any two systems to communicate, even though they might not both be located on the same network. For instance, a UNIX host can **rlogin** over Ethernet to any of the CPU's in a chassis, even though only one of the CPU's in that chassis 'owns' an ethernet controller.

Because multiprocessing and network communications use exactly the same communications facilities, it is extremely easy to configure a systems. Tasks may be moved from CPU to CPE, or moved from an Ethernet connection to a backplane connection, with absolutely no change in application software. Need more compute power? Add another CPU. Need more communications bandwidth? Move to a shared backplane, or a faster network. All that changes is the socket addresses.

Although multiprocessing is an ideal architecture for many applications, development of such systems can be fraught with problems. Though sharing a backplane between a UNIX system and a real-time CPU can work very well, this can also be very dangerous during the development phase. The real-time system might be working with new hardware, receiving interrupts, and accessing hardware with not-yet-debugged code, all of which can make that system very volatile. Doing all this on the same backplane as the UNIX development system, which needs to remain very stable and secure, and might even contain all the source files, is enough to make a system administrator tremble with fear.

The approach taken by VxWorks to solving this problem is simple. During development, the target system should be loosely coupled, over an Ethernet. Because the communications facilities are exactly the same, the target CPU can be moved into the UNIX box later on, when the real-time code is solid and debugged. When this happens, there will be no changes required in the application code, except for new network addresses. This provides the best of all worlds; security during the development phase, and speed and low cost for the final system.

Figure 2: Solutions to Systems Integration Problems

Problem	Solution
Inter-process Communications	TCP/IP sockets RPC (Remote Procedure Calls)
Remote File Access	FTP (File Transfer Protocol) NFS (Network File System)
Program Download Over Network	VxWorks Boot VxWorks Loader
Remote Debugging Interactive Control	rlogin and rsh remote source language debuggers
Multiprocessing	VxWorks backplane network driver

tion to and from a host UNIX system and provide general real-time support, such as memory management, interrupt support, task support and so on. It includes a run-time system, powerful symbolic debugging facilities, and a complete development environment specifically designed to work in partnership with UNIX.

During development, UNIX is used to edit, compile, link and store real-time code that will be run, debugged, and tested in the VxWorks environment. At run-time, one or more VxWorks systems may be networked with one or more UNIX systems on a high-speed network or over a backplane, allowing each to perform the tasks at which it excels. Programs running on either UNIX or in the VxWorks environment may communicate with each other using the normal UNIX socket interface, TCP/IP, and other communications tools (see Table 1 for an explanation of the communications facilities). In addition, a program running in the VxWorks environment is able to use the UNIX systems as virtual file devices. Files on any UNIX system may be accessed, via the network, exactly as if they were local to the VxWorks system. When the two systems are able to communicate smoothly, quickly, and powerfully, the fact that they are different systems running in different boxes can become almost transparent.

An environment such as this which consists of two different but cooperating operating systems can have a number of distinct advantages over using a single operating system for development, real-time use, and non-real-time use. First and foremost, the VxWorks/UNIX partnership allows each to do what it does best; the UNIX system for development and non-time-critical applications, and VxWorks for real-time and non-real-time functions into separate CPU's, each may be optimized for the task at hand. For example, in a multi-user development system running editors, compilers, and other tools, isolation of multiple users for protection is of paramount importance. Unfortunately, providing such protection tends to slow down the task context switch time considerably, which is detrimental in a real-time system. The solution is to use UNIX, which has such protection built in, for development, and to use a system like VxWorks, which is optimised for rapid task context switch, for real-time use.

"A two-OS development environment has a number of distinct advantages over a single OS environment."

Hardware requirements are also very different. The development system requires large amounts of disk storage, provision for a number of terminals, line printers, tape drives, etc. The real-time target system, on the other hand, is likely to be configured very differently.

Separating real-time and non-real-time functions into separate machines has an number of other advantages: Real-time and non-real-time functions don't usually 'live together' very well. Performance of both is likely to be much better if they are separated. During development, real-time systems tend to be very volatile yet the development machine needs to be stable, since a number of programmers might be stored on it. By keeping the system 'loosely coupled' over a network, neither system is able to disrupt the other. Also, the host machine is not affected by the real-time load and tasks may easily be moved around between systems in order to efficiently balance the processing load.

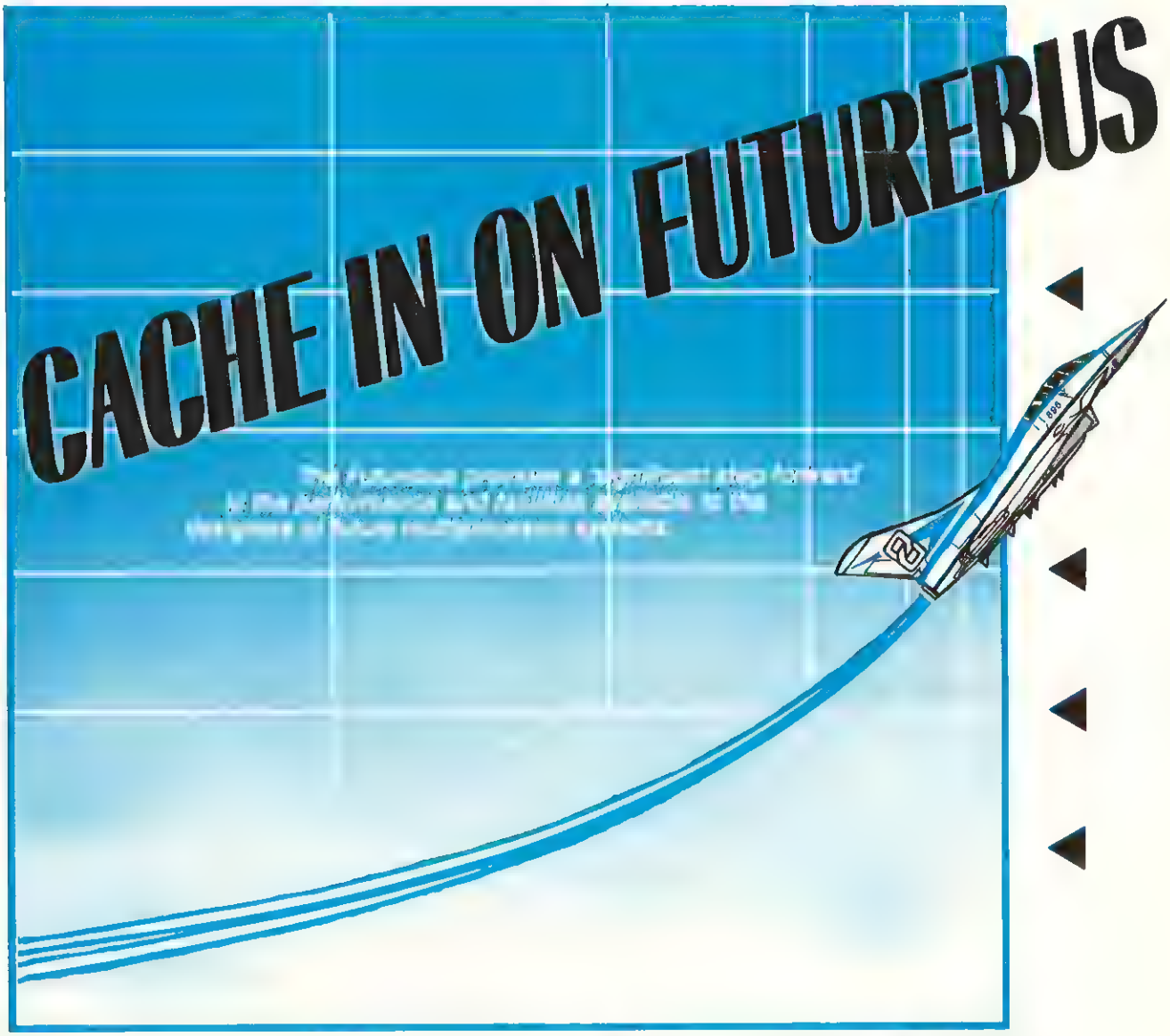
Despite the existence of two different environments, much has been done to make them as common as possible. Thus, VxWorks is designed with UNIX compatibility in mind. This compatibility lies in several areas. VxWorks understands UNIX object file format, and is able to load UNIX object files directly. This means that a program compiled on the UNIX host can be loaded and executed on the real-time system, with no further processing. For instance, the following are commands typed to the VxWorks shell (see Figure 1 for VxWorks block diagram):

```
->ld<host:myFunction
->myFunction1,2,3
->sp myFunction,1,2,3
```

The first command loads the object module **myFunction** from the UNIX host (across the network). The second command calls the function directly, with three parameters. The third command spawns **myFunction** as a separate task.

VxWorks is also source-compatible with UNIX in many areas. It uses the same basic I/O calls (**open, close, read, write, create, delete, ioctl**), the same formatted I/O calls (**printf, scanf**, etc) and the same memory management routines (**malloc, free**, and **realloc**). It also uses the same network calls as BSD 4.3 UNIX. This compatibility makes it possible to run many VxWorks programs directly on UNIX, and also makes it as easy as possible for a

CACHE IN ON FUTUREBUS



87' 88 89 90 91 92 93

- **EXCEPTIONAL PERFORMANCE**
- **ARCHITECTURAL INDEPENDENCE**
  Support for shared memory systems
  (Cache coherency protocol)
  True multiprocessing
  No central elements
- **FAULT TOLERANCE**
  Dual bus support
  Live insertion/withdrawal
- **ELECTRICAL INTEGRITY**
  Solves the "Bus Driving Problem"
- **DESIGN LONGEVITY**
  Technology Independence
  Expanding Silicon Support

CACHE IN ON FUTUREBUS

NAME.....

ADDRESS.....

.....

COMPANY..... TEL: .....

ES/10/87

The response address for the coupon is:
FUTUREBUS INFORMATION SERVICES,
UNIT 2, ROWAN CLOSE,
ST. PETER'S PARK, BRACKLEY,
NORTHANTS, NN13 5UP

Futurebus Manufacturers & Users Group Looking forward to your prompt response.

CIRCLE NO 791

Table 2: Talking Across Networks - UNIX and VxWorks

By providing a fast, convenient connection between the host and target systems, UNIX can be fully utilized as a development system, and as a provider of non-real-time services in a distributed real-time system. The DARPA networking standard, TCP/IP, is an excellent base for all of the required facilities. TCP/IP includes an 'Internet Protocol' (the IP portion of the name) which handles transparent routing of messages between hosts, even across multiple networks, and even if those networks involve different transmission media. TCP/IP is available on all BSD-type UNIX systems v and as an option for most System V versions.

The primary mode of communication is process-to-process sockets. In addition, industry-standard extensions provide a number of 'higher level' modes of communication, such as remote file access, remote procedure calls, remote login, and even remote window and graphics access.

The Internet Protocol is the base network protocol of the Internet protocol family. With IP, each host (ie CPU) in the network has a unique Internet Address for host-to-host message delivery. If multiple networks are connected together, IP keeps track of the interconnections and will route messages from one network to another when necessary. Thus, messages can be sent from a host on one network to a host on another network via IP. All of this routing and internetwork capability is totally transparent to the application.

Although it is possible to access IP directly, almost all applications will use one of the higher-level protocols, such as TCP. While raw IP provides only a single logical connection per host, TCP provide multiple connections, allowing many process-to-process connections within each host. In addition, TCP is flow-controlled, and guarantees that data will be delivered reliably, in the same sequence that it was sent and without duplication.

Sockets are used as a software interface to a variety of network protocols, for interprocess communication. A socket is a communication end point which gets linked to a port within a host (a CPU). Sockets may use TCP/IP (or any other protocol) for sending data between any two ports on any two CPU's on a network

Sockets are analogous to telephones. When a process binds a socket to a port number, it is assigning a telephone number to that socket. Another process on any CPU on the network can create another socket and request a connection to the first process' socket. In other words, one process can 'dial up' the other one by stating the network address and port number it wants to connect to, that is, by 'dialling' a 'telephone number'. The second process can then accept the connection, or 'answer the phone'. Once the connection is established, that is, once both processes are talking on the phone, a virtual circuit is set up between them that allows bi-directional, reliable, error-free socket-to-socket communication.

The VxWorks environment provides normal UNIX socket calls, which allow real-time VxWorks processes and UNIX processes to communicate with each other over a network or a backplane. The VxWorks socket calls are source compatible with BSD4.3 UNIX. Any process can open one or more sockets to which other sockets may be connected. Data written to one socket of a connected pair may be read from the other socket. This network link is totally transparent in the communications. In fact, the two processes don't necessarily know whether they are communicating with another process on the same CPU or another CPU, or with a VxWorks process or a UNIX process.

Communications Tools

Remote Procedure Calls (RPC) is a communication tool that sits on top of the basic sockets. This is the system that allows a program running on one machine to actually make subroutine and function calls to routines running on another computer on the network. Utilities are available which make this almost transparent to the two (host and target) programs, to the extent that they don't need to be aware that they are calling (or being called by) another machine. For many functions, this is a simple way for two programs to communicate than using sockets directly.

Remote file access across the network is naturally also available. A program running on VxWorks is able to use the UNIX systems as 'virtual file devices'. Files on any UNIX system may be accessed, via the network, exactly as if they were local the VxWorks system. For example, **/dk/file** might be a file local to the VxWorks system, while **/host/file** might be a file located on another machine entirely. To a program running under VxWorks, the files operate in exactly the same way; only the name is different.

Another natural extension to the basic network capabilities is the ability of any computer to log in to another computer on the net. This allows a user on one machine to have access to other connected machines, or for a program to execute commands on another machine. There are two standard protocols use for this; **rlogin** and **telnet**. Both are available in VxWorks. Thus, a user on a UNIX system can **rlogin** to a VxWorks system, or vice versa.

One of the newest tools available to systems architects is 'network-transparent' window systems. With such a system a process is able to open a window for input and output, and graphics display, on another computer via a network. One such system that is becoming an industry-wide standard is the X system, developed at MIT. VxWorks will support the X-window system. This will allow real-time tasks to have graphics capabilities, even if they do not physically have any graphics hardware of their own.

At run-time, real-time VxWorks systems and UNIX systems can share a network or a backplane, allowing each to perform appropriate tasks, allowing each to perform appropriate tasks. Large real-time systems often require non real-time components for which UNIX is non-real-time components for which UNIX is perfectly suited. The ease of communications provided by the VxWorks-UNIX network, TCP/IP, and extensions like RPC and X-windows make it very convenient best suited to it. In this sort of configuration, the UNIX system can either be thought of as providing services to the real-time systems (as a file server, or data-base server, for instance), or as a high level controller of a network of real-time systems.

UNIX programmer to learn to work with the real-time portion of the system.

Using VxWorks

During development, all code is stored, edited, compiled, linked, etc on the UNIX host. The programmer has access to all the tools provided by the UNIX system to help with this. Once compiled, programs are quickly and easily loaded by the VxWorks target system, from the UNIX host, via the remote file system. VxWorks provides tools that allow the program to be tested and debugged while running on the target system.

The programmer's primary interface to VxWorks is via the VxWorks shell. Most traditional small systems are supplied with some kind of command interpreter program which provides the user with a limited and fixed set of commands that can be invoked interactively. The VxWorks shell, however, provides a much simpler and vastly more powerful mechanism: the shell can interpret and execute almost all expressions of the C language, including calls to functions, and references to variables, whose names are found in the system symbol table. Thus the shell can be used to call VxWorks system functions, to call any application functions, to examine and set application variables, and even as a general purpose calculator with all C operators.

From the VxWorks shell, the programmer can call any subroutine in the system or in his application. He can spawn any subroutine as a task, or connect any subroutine to an interrupt or watchdog timer. At any time, he may perform a stack trace on any task. Any tasks may be suspended, resumed, or changed in priority. He can insert a breakpoint in any task, or for all tasks, or single step any task. He can time any function or group of functions in several ways, or monitor CPU utilization by task.

The programmer may communicate with the VxWorks shell on a terminal or directly from the UNIX host via **rlogin** or **telnet**. On a UNIX workstation, the programmer can even open window that communicates with the VxWorks shell. By opening such windows, the programmer can monitor and control one or more real-time VxWorks systems right from his desk.

Source language debugging is also available. The debugger runs on the UNIX host, but allows debugging of target processes by interacting, via the network, with a special debug process running on the target processor. Thus, the programmer may use the same high-level language debugging tools to debug his real-time processes as he would to debug native UNIX programs.

The VxWorks system provides a number of additional facilities to speed development, such as:

Network Boot. VxWorks provides boot ROMs for all supported target-CPUs that allow automatic booting over a network or a backplane.

Shareable Code. All code can be shared between multiple tasks. The same routine can even be spawned as multiple tasks, all running the same code, but with their own stacks.

Load-Time Linkage. In order to facilitate code shareing, the VxWorks loader resolves undefined externals when a module is loaded. When a task calls **printf**, for instance, it uses the same copy of the **printf** code that other tasks use.

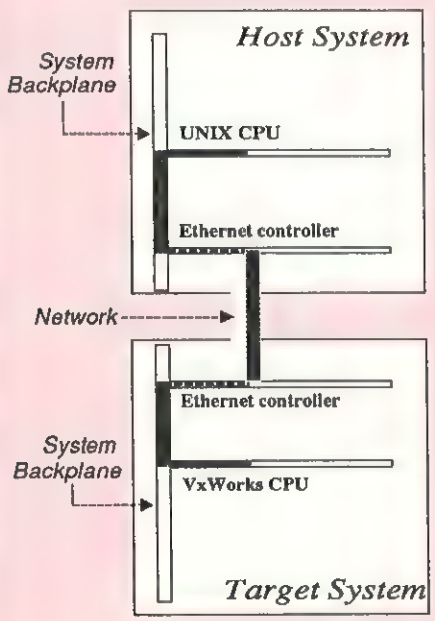
Availability of Utility Routines. VxWorks includes over three hundred fifty subroutines, many of which are utility functions, such as linked list and ring buffer manipulation, and optimized buffer moving and copying. These are often routines that were needed by the system, so they have been packaged and documented and are available to application code.

Performance Monitoring Tools. Tools are provided to time any function or sequence of functions,and to monitor CPU utilization by various tasks and interrupt level code.

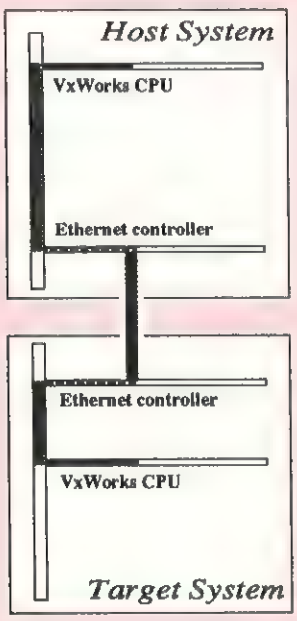
Tools for Romming. VxWorks provides all the tools necessary for putting a system, including the application, in ROM. No software changes are necessary; VxWorks will automatically copy data, as necessary, between ROM and RAM at initialisation time.

Kirby, Fiddler and Wilner are with Wind River Systems of Emeryville, California who manufacture VxWorks. They can be contacted on 0101 415 428 2623.

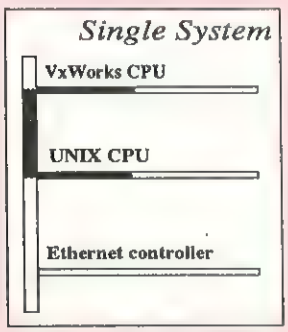
Figure 3: Configurations of VxWorks and UNIX



1. *UNIX-VxWorks over network*



2. *VxWorks-VxWorks over network*



3. *UNIX-VxWorks across backplane*

Table 3: Sockets Explained

The Socket Schema

Inter machine connection and communication is not a new concept in UNIX. However, a completely smooth, portable and transparent communication is a reasonably recent introduction. Networked UNIX poses a number of problems, not least of which is when one process wishes to communicate to another, sending data and specific addresses of data. This type of information should be packaged according to the specific network and protocol being used. Naturally, any communication system which requires a machine specific modification goes against the 'openness', of UNIX and greatly hampers portability. The UNIX solution is 'Streams'. The solution offered by Berkeley in BSD UNIX is the 'sockets' system. Sockets operate by introducing a new layer into the programming model and new system calls to accompany this.

The Jargon

Communication is broadly classified into two types of domain. Communications between processes on one machine are in the **UNIX system domain**, and when they communicate between different machines they are in the **Internet domain**. Protocols established by the US Defense Advanced Research Project Agency (DARPA) are generally used for this communication and the IP protocol is now supported by most UNIX workstations.

A socket can be one of two types: a **datagram** or a **virtual circuit**. The virtual circuit allows gives a means of controlling transmission and receipt, while datagram messages are inherently unreliable. They give no indication that transmission has occurred accurately, or at all. Their advantage is that they are relatively easy to set up and so generally become the 'cheap and cheerful' way of communicating. Two popular communications protocols are used in these two types of socket, the Transport Connect Protocol (TCP) for virtual circuits and the User Datagram Protocol (UDP) for datagrams. There can be a varied combination of domain and protocol, though the obvious preferences are TCP and IP.

Socket Communications

First, a client process must create a socket using:

```
sd=socket(format, type, protocol);
```

No communications links are established at this time. One system call makes the socket and creates a socket descriptor. A second call then associates a name with the descriptor:

```
bind(sd, address, length);
```

The kernel then connects the socket using *connect*. The server process must also establish a working socket. It too, must create a socket descriptor, give it a name and connect it. The server can then issue a *listen* call which establishes a queue for the receipt of messages. An *accept* call then receives incoming requests for a connection to a server process.

The **send** and **recv** calls are then used by the client and server to communicate across a connected socket. The calls take the following format:

```
send(sd, msg, length, flags);
recv(sd, buf, length, flags);
```

where *sd* is the socket descriptor, *msg* is a pointer to the data being sent (and *buf* is a pointer to an array to receive this data on the server), *length* is the length of sent data (or the expected length of received data). *flags* can be sent to send data which is not to be considered as part of the regular sequence of data exchange, as may be required with a remote login program. The more usual **read** and **write** system calls can also be used with sockets as if they were normal files.

Example in the UNIX System Domain

The two code listings show how a server and client program may use the socket system calls to communicate.

Listing 1: Server Process

```
#include <sys/types.h>
#include <sys/socket.h>

main()
{
  int sd, ns;
  char buf[256];
  struct sockaddr sockaddr;
  int fromlen;

  sd = socket(AF_UNIX, SOCK_STREAM, 0);

  bind(sd, "sockname", sizeof("sockname")-1);
  listen(sd,1);

  for (;;)
  {

    ns=accept(sd, &sockaddr, &fromlen);
    if(fork()==0)
    {
      close(sd);
      read(ns, buf, sizeof(buf));
      printf("server read '%s'n", buf);
      exit();
    }
    close(ns);
  }
}
```

Listing 2: Client Process

```
#include <sys/types.h>
#include <sys/socket.h>

main()
{
  int sd, ns;
  char buf[256];
  struct sockaddr sockaddr;
  int fromlen;

  sd = socket(AF_UNIX, SOCK_STREAM, 0);

  if(connect(sd, "sockname",
sizeof("sockname")-1)==-1)
    exit();

  write(sd, "hello!", 6);
}
```

Microsoft Authorised

QA=MS OS/2

The First Authorised Training Course

"The reason we gave QA our authorisation for this course was the unqualified success of the QA/Microsoft Windows courses. We can rely on QA instructors to give software developers a thorough understanding of our new and extremely powerful operating system."

Phil Buggins,
Technical Director of Microsoft Ltd.

Other QA courses:

- Windows Programming
- 'C' Programming under DOS
- 'C' Programming Standards
- Advanced 'C' Software Development
- 286/386 Systems Programming
- PC Networks
- Xenix Technical Support
- PC Maintenance and Repair
- 8088/8086 PC Assembler

QA Training Ltd.,
Cecily Hill Castle,
Cirencester,
Glos. GL7 2EF
Tel. (0285) 5888

QA TRAINING

*MS OS/2 is a trademark of Microsoft Corporation

CIRCLE NO 792

Running a NETBIOS Session with INT 5CH

Russell Lloyd looks at using the PC NETBIOS interface to write network software and works through a full example.

This article gives details of the NETBIOS interface and how it can best be exploited when writing software to run over PC networks. Firstly, I'll look at the general functions offered and then look at how the 5CH interrupt and the network control block (NCB) work together in practice. I conclude with a worked example in C showing how all the parts fit together.

Despite the general lack of standards in the field of Local Area Networks, one de facto standard that has emerged is the 'NETBIOS' software interface standard. It was originally defined by IBM and provides access to network specific functions that are not available through the MS-DOS operating system. In much the same way as the IBM ROM BIOS enables software developers to write software that makes use of the IBM hardware, the NETBIOS has encouraged the development of network-specific software such as gateways, bridges, added value servers, mail-systems etc.

Virtually every network vendor now provides a NETBIOS compatible network interface. Allowing NETBIOS specific application programs to communicate across the LAN. Software written to utilise the NETBIOS interface will generally be able to function on any LAN with a truly compatible NETBIOS interface.

NETBIOS Functions

NETBIOS provides access to four groups of functions: Datagram Services, Session Services, Name Service functions and some general functions. These general functions consist of some functions such as Adapter Reset, Local/Remote Hardware Statistic and Unlink (Remote Boot Facility), which are somewhat hardware specific and may be emulated to varying degrees by different LAN vendors. I will now examine the other three groups of functions in order.

Name Service functions are important and deserve further explanation. Basically they concern the naming and grouping of workstations on the network. Any station on the LAN can arrange to be known by up to 16 different names at any time, in addition to the physical adapter card address, called the Permanent Node name. These can be unique names or group names, whereby a group of stations can be addressed together by the chosen name. It is therefore possible to write network application software that uses logical names for network addressing rather than physical addresses, which makes for hardware independent software. Group names can be used to identify stations that are performing similar functions where, for example, a group of stations might be executing a program within an accounting department of a corporate LAN, for example. The ability of NETBIOS to accept up to 16 names makes it possible for multiple independent application programs to use the NETBIOS concurrently.

Datagram Support

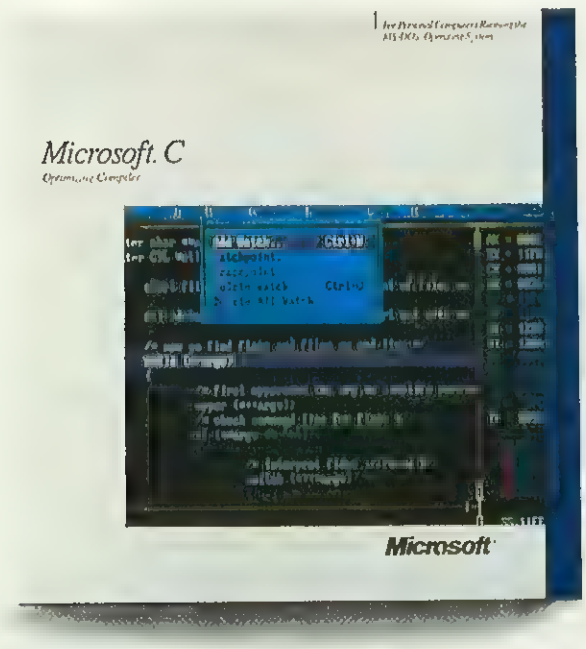
The datagram support offered by NETBIOS is often used by simple network applications, most notably those ported to NETBIOS from other network interfaces. There is a slight disadvantage with the datagram approach: they have no way inherent way to guarantee successful delivery to the destination. As a result, datagrams require the application program to take error detection and recovery into account. A datagram is like a letter – it is sent to a particular address, but unless the recipient writes back, there is no way of knowing whether the letter arrived. In the IBM NETBIOS, there is also no buffering of datagrams by the receiver. So if the receiving application program was not ready to receive the datagram it is discarded. At least with a letter it is still on the mat when you return from holiday!

Facilities exist within the NETBIOS to send and receive datagrams to a unique

```
struct NCB
 {
   char NCB_COMMAND;
   char NCB_RETCODE;
   char NCB_LSN;
   char NCB_NUM;
   char far *NCB_BUFFER;
   int NCB_LENGTH;
   char NCB_CALLNAME[16];
   char NCB_NAME[16];
   char NCB_RTO;
   char NCB_STO;
   int  (far *NCB_POST)();
   char NCB_LANA_NUM;
   char NCB_CMD_CPLT;
   char NCB_RESERVE[14];
 };
```

Figure 1: The format of an NCB - the NCB.H file.

C5.0 has three features professional programmers can't live without.



Speed.

Fast Execution Speed.

	Microsoft® C 4.0	Microsoft C 5.0
Sieve (25 iterations)	5.7	3.3
Loop	11.0	0.0*
Float	19.9	0.1
Dhrystone	22.8	19.1
Pointer	14.2	7.4

- New optimizations generate the fastest code:
  - —Inline code generation. **NEW!**
  - —Loop optimizations: **NEW!**
    - —Loop invariant expression removal. **NEW!**
    - —Automatic register allocation of variables. **NEW!**
  - —Elimination of common sub expressions.
  - —Improved constant folding and value propagation.
- Fine tune your programs for even greater speed:
  - —Coding techniques for writing the fastest possible programs are included in the documentation. **NEW!**
  - —Segment Allocation Control:
    - —Group functions into the same segment to get faster NEAR calls. **NEW!**
    - —Specify which segments receive variables to yield faster NEAR references. **NEW!**
  - —Uses register variable declarations.
  - —Mix memory models using NEAR, FAR & HUGE pointers.

Benchmarks run on an IBM® Personal System/2™ *Time is negligible

Speed.

Fast Compilation.
Fast Prototyping.

Microsoft C Version 5.0 includes QuickC,™ which lets you edit, compile, debug, and execute in an integrated environment. It's ideal for prototyping.

- In-memory compilation at over 10,000 lines/minute. **NEW!**
- Built-in editor with parentheses, bracket and brace matching.
- Use the integrated debugger to animate through your program, add watch variables and set dynamic breakpoints. **NEW!**
- MAKE file is automatically generated for you. Simply indicate the modules you want to use, then MAKE recompiles and links only those modules that have changed. **NEW!**
- Full C 5.0 compatibility:
  - —Completely source and object code compatible.
  - —Emits CodeView®-supported executables.
  - —Identical compile/link command line switches.

And speed.

Fast Debugging.

Microsoft C Version 5.0 includes Microsoft CodeView, our source-level windowing debugger that lets you debug more quickly and thoroughly than ever before.

- Debug larger programs:
  - —Debug through overlays created by the Microsoft overlay linker. **NEW!**
  - —Expanded Memory Specification (EMS) support. **NEW!**
- Fast debugging through precise control of your program execution:
  - —Access source level and symbolic debug information from your Microsoft C, FORTRAN, and Macro Assembler programs. **NEW!**
  - —View your source code and assembly simultaneously.
  - —Watch the value of variables change as you execute.
  - —Set conditional breakpoints.
  - —Animate or single step through your program.
- CodeView brings you as close as you've ever been to your hardware:
  - —Swap between your code and output screens.
  - —Watch your registers and flags change as your program executes.

Microsoft®

C 5.0 will be available soon. If you purchased Microsoft C 4.0 after August 1st, 1987, we'll give you a C 5.0 upgrade. Free. For your free information packet, call:

(0734) 500741

Figure 2: Possible NETBIOS Commands

```
/*              NETBIOS.H                */

#define Netbios_int                   0x5c
#define max_NCBs                        10
#define max_packets                    200
#define maxbuff                       2048
#define delay                            5

#define wait                             0
#define no_wait                       0x80

/* Poss. values of NCB_COMMAND byte */

#define NCB_reset                     0x32
#define NCB_adaptor_status            0x33
#define NCB_cancel                    0x35
#define NCB_unlink                    0x70

#define NCB_send_datagram             0x20
#define NCB_receive_datagram          0x21
#define NCB_send_broadcast            0x22
#define NCB_receive_broadcast         0x23

#define NCB_call                      0x10
#define NCB_listen                    0x11
#define NCB_hangup                    0x12

#define NCB_add_name                  0x30
#define NCB_delete_name               0x31
#define NCB_add_group_name            0x36

#define NCB_send                      0x14
#define NCB_recv                      0x15
#define NCB_receive_any               0x16
#define NCB_chain_send                0x17
```

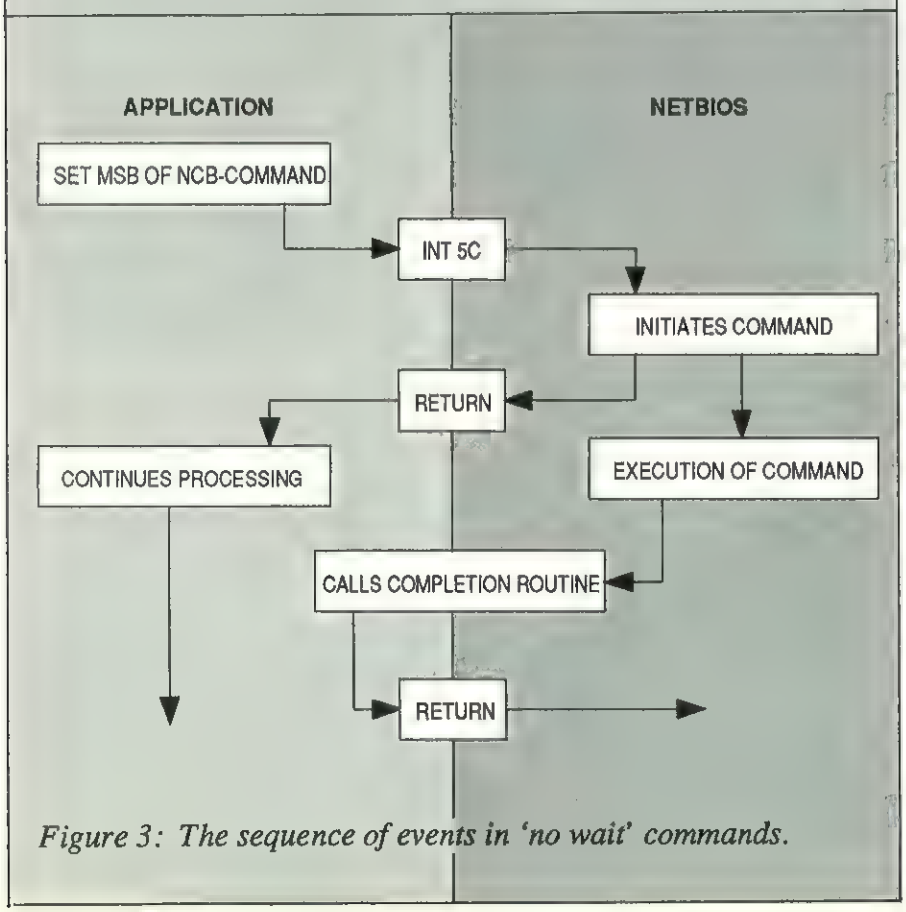


Figure 3: The sequence of events in 'no wait' commands.

name address, a group of stations having the same group name or two all stations on the LAN.

Session Support

This is the most important facility provided by the NETBIOS for 'interstation' communication. A session is a logical communication channel set up between two names on the LAN, which normally correspond to application programs. Whilst a session is in progress, the NETBIOS takes care of error detection and recovery, buffering and flow control. A session is like a telephone call, it has to be explicitly set up and answered and is ended (using **CALL**, **LISTEN** and **HANGUP** commands), and during the session, two-way conversation is possible with **SEND** and **RECEIVE** commands. If at any time the line drops, or the station cannot continue to support the call, the other station is informed. Any serious application program should use the NETBIOS session support facilities to guarantee a reliable 'conversation'.

An application program can have many sessions outstanding to different destinations simultaneously, and is a common way of implementing file server software.

The session commands are recommended because they guarantee delivery and remove the need for high level application protocols, which are usually necessary in programs trying to use datagrams for reliable transfer. Miscellaneous session commands are included to initialise the adaptor, gather network statistics etc and are not discussed further in this article.

The INT 5C Interrupt

Programs wishing to make use of the NETBIOS facilities have to adhere to strictly defined interface standards. Every NETBIOS function is initiated by constructing a Network Control Block (NCB) which contains all of the information the NETBIOS needs to execute the command. Every NCB consists of a 64 byte block of memory shown as a C structure in Figure 1.

Every time a command is initiated, the NCB must be filled in appropriately. Different fields apply to different commands, but the **NCB-COMMAND** field always informs the NETBIOS which type of command has to be performed. The possible NETBIOS commands are listed in the #include file **NETBIOS.H** in Figure 2.

Commands are issued to the NETBIOS by supplying the address of the Network Control Block (NCB) in the ES:BX register pair and then issuing interrupt INT 5CH.

Figure 4: Main Network Program Procedure

```
/*----------------MAIN----------------*/

main (argc,argv)
int argc;
char **argv;

{
float packets_per_sec,K_per_sec,test_time;
int   packet_total;

/* get no. stations, and max req. packet size ( 1 - 2048 ) */

   printf("\nEnter number of stations to test (1 - 9) ..");
   scanf("%d",&station_total);
   printf ("\nEnter the packet size ( 0 - %4d ) ..",maxbuff);
   scanf ("%d",&packet_size);
   printf("Enter your station number ( 1 - 9) ..");
   scanf("%d",&station_number);

/*  Initialise the NCBs etc.  Attempt to add the unique name
    'STATIONx' to the network, where 'x' is the value given
    by station_number, and the group name 'TESTGROUP'.  */

initialise();
add_name (station_number);
add_group_name ();

/* if we are station 1, we become the 'server' station.*/

if (station_number == 1) initiate_packets();
    else respond_to_packets();

/* print the throughput statistics and exit */

time(&end_time);
if (station_number == 1) packet_total = (station_total-1)
                                        *max_packets;
   else packet_total = max_packets;
printf("\nSent %4d packets ",packet_total);
printf("\nReceived %4d packets",packet_total);
test_time = end_time - start time;
printf("\nTime taken was %4.2f seconds",test_time);
packets_per_sec = (2*packet_total)/test_time;
K_per_sec = (packets_per_sec*packet_size)/1024.0;
printf("\nPerformance of this machine was ..");
printf("\n\n     %4.2f packets per second" ,packets_per_sec);
printf("\n      %4.2f K per second",K_per_sec);
exit(0);
}
```

NETBIOS will return control back to the initiating program when the command has fully completed.

There are some interesting variations on this, however, such as the 'no-wait' or asynchronous option. This returns control immediately to the calling program and continues processing the command in parallel. The no-wait option is achieved by setting the most significant bit of the **NCB-COMMAND** byte: NETBIOS then accepts the command and returns control back to the application program before the command has completed.

This asynchronous task then runs to completion and will usually want to react with the calling program again. So how does the application program know when the command has completed? There are two ways. Firstly, the application can check the **NCB-CMD-CPLT** field in the NCB as the NETBIOS will change the **NCB-CMD-CPLT** field from FF Hex to a completion code value when the command has completed. The application program can therefore periodically check this byte to determine when a command is finished. A more complex method is to specify the memory address of a completion handler (also called a post address or completion routine) in the **NCB-POST field** of the NCB when the command is initiated. When the command has completed, the NETBIOS calls the address (as if it were an interrupt handler), asynchronously to the rest of the program execution. The completion handler can, therefore, perform some application-specific processing at the time when the command completes. The return code is placed in the **NCB-RETCODE** field of the NCB. This mechanism is useful in a multi-tasking environment where the completion routine can be used as an 'event' to reschedule a suspended task. Figure 3 shows the sequence of events in this no-wait command process. Using the 'no wait' form of network command is essential in server environment where further commands can be processed whilst waiting for a network command to complete.

Every command has a number of command-specific return codes associated with it. There is a set of immediate return codes, one of which is returned in the AL register which indicates whether the command was initiated successfully. The final status of the command is placed either in the **NCB-RETCODE** field of the NCB (for wait style commands) or in the **NCB-CMD-CPLT** field of the NCB (for no-wait style commands). A zero return code indicates successful command completion.

Establishing A Session

Let us briefly look at how to set up a session and pass data between two programs – the sender and receiver. The sender (or caller) program first formats an NCB with its name and the name of the destination station by filling in the **NCB-NAME** and **NCB-CALLNAME** fields. The **NCB-COMMAND** byte is then sent to 10H or 90H (to execute a no-wait call command) and the NCB issued to the NETBIOS using the INT 5CH interrupt. The receiver must issue a corresponding **LISTEN** command, quoting the same names. When the call completes, the **NCB-LSN** (local session number) field will have been filled in by the NETBIOS in each NCB. This field identifies the session to the NETBIOS which may have several others outstanding, and must be quoted in every successive command between the two stations for that particular session. Each side of the session can then initiate send and receive commands, with the NETBIOS taking care of error recovery and buffering until one or both parties decide to terminate the call by using a NETBIOS **HANGUP** command.

The sample program in this article written for the Microsoft C compiler, demonstrates a number of possible NETBIOS commands. In this program, several stations are configured to belong to a group of stations known by the group name TESTGROUP. One station assumes the role of the server station, the others act as responders. The server generates a fixed number of messages to each responder, each of which echoes every received message back to the server. The server calculates how long it took to send and receive this fixed number of messages and therefore is able to give an indication of the efficiency of the NETBIOS emulation and the underlying network.

Figure 5: Structures and Initialisation

```
#include "stdio,h"
#include "dos.h"
#include "string.h"
#include "time.h"
#include "NCB.h"
#include "netbios.h"

#define true 1
#define false 0

int tx_count[max_NCBs];
int rx_count[max_NCBs];

struct NCB send_NCBs[max_NCBs];
struct NCB recv_NCBs[max_NCBs];
struct NCB an_NCB;

char tx_buffer[maxbuff];
char rx_buffer[maxbuff];

int packet_size, station_total,
                        station_number;
long start_time, end_time;

initialise()
{
int i;

/*  Initialise all send NCBs to point to
    the same data buffer.  Initialise all
    receive NCBs to point to the same
    receive buffer.  This data can there-
    fore never be guaranteed to be valid
    at any time, but we don't look in it
    anyway.  */

   an_NCB.NCB_BUFFER = rx_buffer;
   an_NCB.NCB_LENGTH = 512;
   for ( i=0 ; i< max_NCBs ; i++ )
   {
      send_NCBs[i].NCB_BUFFER = tx_buffer;
      send_NCBs[i].NCB_LENGTH = packet_size;
      recv_NCBs[i].NCB_LENGTH = packet_size;
      recv_NCBs[i].NCB_BUFFER = rx_buffer;
   }
}
```

Figure 6: Utility Subroutines

```
/*-----------SEND and RECEIVE DATAGRAM----------*/

send_datagram()
{
  an_NCB.NCB_COMMAND = NCB_send_datagram;
  strncpy ( an_NCB.NCB_CALLNAME , "TESTGROUP", 9);
  issue_NCB(&an_NCB,wait);
}

receive_datagram()
{

  an_NCB.NCB_COMMAND = NCB_receive_datagram;
  issue_NCB(&an_NCB, wait);
}


/*-------------LISTEN and CALL-----------------*/

listen(NCB)
struct NCB *NCB;
{
  NCB->NCB_COMMAND = NCB_listen;
  strncpy( NCB->NCB_NAME , "STATIONx", 8);
  NCB->NCB_NAME[7] = station_number + 0x30;
  strncpy( NCB->NCB_CALLNAME , "STATION1", 8);
  issue_NCB( NCB, wait);
}

call(NCB, station)
struct NCB *NCB;
int station;
{
  NCB_>NCB_COMMAND = NCB_call;
  strncpy( NCB->NCB_CALLNAME , "STATIONx", 8);
  NCB->NCB_CALLNAME[7] = station + 0x30;
  strncpy( NCB->NCB_NAME , "STATION1", 8);
  issue_NCB (NCB, wait);
}


/*--------ADD NAME and ADD GROUP NAME----------*/

add_name(station)
int station;
{

  an_NCB.NCB_COMMAND = NCB_add_name;
  strncpy( an_NCB.NCB_NAME, "STATIONx", 8);
  an_NCB.NCB_NAME[7] = station + 0x30;
  issue_NCB(&anNCB, wait);
}

add_group_name()
{

  an_NCB.NCB_COMMAND = NCB_add_group_name;
  strncpy( an_NCB.NCB_NAME, "TESTGROUP", 9);
  issue_NCB(&an_NCB, wait);
}
```

The sample program is split into several sections. The C main routine performs all user interaction and executes the **initiate-packets** or **respond to packets** subroutine depending on whether the station is to be the server (station 1) or a responder. This is shown in figure 4.

In order to use the NETBIOS with many concurrent sessions, there must be an NCB for every possible outstanding command. No NCB can be reused whilst a command is in progress. To achieve this, the data structures shown in Figure 5 declare two arrays of NCBs, each array element being a data block of the structure type NCB. There is a send NCB and a receive NCB allocated for each of the possible stations involved in the test.

So that the program can keep track of the number of messages sent and received to each responder, two further arrays are defined, namely **tx-count** and **rx-count**. These contain a value corresponding to the number of messages sent to and received back from to every possible responder.

There are a number of assorted subroutines shown in Figure 6, all of which perform a specific NETBIOS function. **Add-name** and **Add-group-name** prepare an NCB by filling in the **NCB-NAME** and **NCB-CALLNAME** fields and then issuing the NCB by calling the **Issue-NCB** subroutine. The unique name formed by the **Add-name** subroutine is Stationx, where x is the station number input by the user in **main**. Once these subroutines have been executed, every station in the test can be accessed by the group name TESTGROUP or the individual station name STATIONx.

The subroutine responsible for the actual communication with the NETBIOS interface is **Issue-NCB** shown in Figure 7. This subroutine can issue a wait or no-wait NETBIOS command using the Microsoft C library functions **segread** and **int86x** to pass the address of the relevant NCB to the NETBIOS interface in the ES:BX register pair. Also shown in Figure 7 is the **Check-NCB-and-issue** subroutine which is used with no-wait NETBIOS commands. If the command has completed (by virtue of the **NCB-CMD-CPLT** field in the NCB no longer being set to FFH) another command is issued until the count parameter indicates that the maximum number of commands have been issued.

Figure 7: Netbios interface routines

```
/*--------CHECK NCB and ISSUE a new one--------*/

int check_NCB_and_issue(NCB, count, result, cmd)
struct NCB *NCB;
int *count;
int *result;
int cmd;

{
int error;

/* Check if send or receive NCB has completed. If
so, issue another unless the max no. has been
issued */

 *result = true;
 if (( NCB->NCB_CMD_CPLT & 0xFF) != 0xff )
 {

    if (*count != max_packets)
    {

       NCB->NCB_COMMAND = cmd;
       issue_NCB (NCB, no_wait);
       (*count)++;
    }
    else *result = false;
 }
}

/*---------------Issue_NCB-------------------*/

int issue_NCB(NCB, wait_option)
struct NCB *NCB;
int wait_option;
{
struct SREGS segregs;
union REGS inregs;
union REGS outregs;
int result;

 NCB->NCB_COMMAND = NCB->NCB_COMMAND +
                    wait_option;
 segread (&segregs);
 inregs.x.bx = (unsigned int) NCB;
 segregs.es = segregs.ds;
 result = (int86x(Netbios_int,&inregs,&outregs,
                                 &segregs)) & 0xff;
if (result) printf ("\nError initiating cmd -
                        result = %2x", result );
return result;
}
```

Figure 8: Server and Responder Code

```
/*----------SERVER CODE-----------------------*/

initiate_packets() {
int i , more , done;
/* Try to establish session with each station.*/
  printf("Initiating packet transmission");
  for ( i = 2; i <= station_total ; i++){
   call( &send NCBs[i], i );
   recv_NCBs[i].NCB_LSN = send_NCBs[i].NCB_LSN;
   printf("\nCALL made with STATION%c",i+0x30);  }

/* delay to let stations Receive Datagram.  Send
group Datagram to synchronise */
time(&start_time);
do time(&end_time);
while ((end_time - start_time)<= delay);
send datagram();
time(&start_time);

/* continually send packet to each station. Don't
send another to it until it has sent a reply until
'max-packets' sent and received from all*/
  do {
     more = false
     for ( i = 2 ; i <= station_total ; i++ )
     {
check_NCB_and_issue(&send_NCBs[i], &tx_count[i],
                                    &done, NCB_send);
check_NCB_and_issue(&recv_NCBs[i], &rx_count[i],
                                    &done,NCB_recv);
     more = more | done; } )
while (more);
}

/*-------------RESPONDER STATION CODE----------*/

respond_to_packets () {

/* First wait for Call to come from Server then
datagram to synchronise with other stations, then
start receiving and echoing data packets. */

listen( &recv_NCBs[station_number] );
printf("\nCall established with Station 1\n")
send_NCBs[station_number].NCB_LSN =recv_NCBs
                          [station_number].NCB_LSN;
receive_datagram();
recv_NCBs[station_number].NCB_COMMAND = NCB_recv;
send_NCBs[station_number].NCB_COMMAND = NCB_send;
time(&start_time);

do {
  issue_NCB( &recv_NCBs[station_number], wait);
  issue_NCB( &send_NCBs[station_number], wait);
  (tx_count[station_number])++;
  } while (tx_count[station_number]<max_packets);
}
```

Figure 8 shows the server specific subroutine **Initiate-packets** which controls the execution of the test, and the responder-specific subroutine **Respond-to-packets**. In order to establish a session with every responder, the subroutine issues a NETBIOS **CALL** command to each of the responders in turn by executing the **call** routine. This formats an NCB quoting the unique station name of the responder station (STATION1, STATION2 etc). The responder is expecting a **CALL** to be set up from STATION1 and has correspondingly set up a **LISTEN** command.

Having successfully established a session with each of the responders, the server code synchronises all the stations before sending a datagram message to the group name TESTGROUP. Because all of the responders are also known by this name, they should all receive this message virtually simultaneously and all record the time of its receipt. Note that this particular application program does not have any way of knowing whether the datagram arrived at all the responders and in fact the program will fail if it doesn't.

Once the stations are all synchronised, the **Initiate-packets** subroutine takes control and performs data transmissions to each responder, waiting for each send command to complete before sending the next. At the same time, the server keeps a receive command outstanding per session to receive data packets that have been echoed back from the responders. In order to maintain maximum throughput on all sessions, the no-wait option is used on all of these sends and receives and the **NCB-CMD-CPLT** field in the NCB is checked to determine when the command has completed. This function is performed by the **Check-NCB-and-issue** subroutine. When a fixed number of packets have been transmitted and received from every responder, the test stops.

The processing performed by each responder is straightforward and is shown in Figure 8 in the **Respond-to-packets** subroutine. Having added the unique and group NETBIOS names, the responder issues a **LISTEN** command (through the **Listen** subroutine) to accept the subsequent **CALL** from the server station. Once done, the responder issues a **RECEIVE DATAGRAM** command to catch the synchronisation datagram that the sender sends to each responder in the group defined by TESTGROUP.

It has not been possible in this article to fully explore the NETBIOS interface. Anyone wishing to learn more should obtain a copy of the IBM PC Network Technical Reference Manual which gives a detailed description.

Russell Lloyd is latterly Product Development Manager for AST Europe Ltd.

Sycero dB from System C the best thing on the programmer's menu

Sycero dB ***The dBase Generator*** can develop both simple and complex programs with greatly reduced development and debugging times. Generated programs have a common user interface and are automatically documented to a high standard.

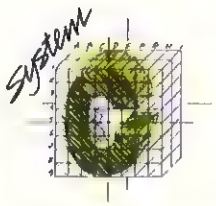
The generator automatically programs standard functions such as menus, posting, file maintenance and reporting. Screens and reports are painted, dBase a new extended language can be used to enhance these standard functions.

Up to ten databases and ten screens can be used in each program and up to 100 lines of code can be added to each field on a report or screen.

Sycero includes a powerful report generator which supports range checks, sort sequence selection, wild card matching, exception reporting, ten levels of subtotal and up to nine secondary report loops per report. It is simple to generate lists, forms and invoice/statement type reports.

When used with Clipper, the generator can automatically compile an entire suite of programs into one. EXE file using overlays.

If you want to stay ahead in dBase and still have time for lunch, order Sycero dB today by filling in the coupon below.



The dBase Generator

System C Ltd 7 Mill Street Maidstone Kent ME15 6XW TEL 0622 55142

EXE 11 87

To: System C Ltd 7 Mill Street Maidstone Kent ME15 6XW
Please Supply _____ copies of SYCERO dB @ £595 + VAT (£684.25) or dB net @ £745 + VAT (£856.75)
I Enclose a cheque for £ _____
Contact Name _____
Company _____
Address _____
_____
Tel _____

CIRCLE NO. 794

Clipper is a registered trade mark of Nantucket. dBase is a registered trade mark of Ashton Tate

A Graphics Toolbox for Turbo C

Building a toolbox requires mastery of interrupts, library creation and linking, recognition of graphics adaptors and quite a lot of maths. In the first article of a two-part series, Kent Porter reveals half.

That Borland's Turbo C comes equipped with 350 functions and macros is pretty impressive until you realise that not one of them has anything to do with graphics. To fill this void, I've constructed my own version of a graphics toolkit for Turbo C.

To make the information more easily digestible, I've divided the project into two bite-sized articles. This piece introduces a basic library of graphics calls and shows how to put them to work in pixel-oriented graphics. Part 2 (to appear in January's .EXE) will combine this graphics library with data structures and C functions to control pop-down menus, dialog boxes and other appearance features that users have come to expect in contemporary software.

The MS-DOS Underpinning

IBM PCs or equivalents rely on the ROM BIOS interrupt 10H (16 decimal) for their graphics capabilities. This interrupt furnishes 'generalised' functions for various aspects of text and all-points addressable (APA) or pixel-oriented graphics. I say generalised because the ROM BIOS accommodates numerous modes, some of which may not be available on certain hardware configurations.

There are 16 core functions in the ROM BIOS video service. These functions govern display aspects such as the position or shape of the cursor, the video mode (text and APA options), the active display page, character output, pixel graphics and others.

The ROM BIOS functions are not known for their speed. In general, they're adequate in text operations but less than adequate in APA graphics. You could devise much faster pixel manipulations by writing to the display memory directly, and indeed, many commercial packages do just that. The speed advantages of this approach, however, are obtained with a high risk of incompatibility: environments such as Windows and OS/2 are much more intolerant of direct display memory access.

In selecting the ROM BIOS trade-off rather than directly writing to screen memory, I've deliberately opted for the more conservative approach on two grounds. First, the ROM BIOS calls are the approved method for doing graphics and future machines will probably support them within the definition of well-behaved. Second, faster processors will cancel out their slower speed, resulting in a zero loss/gain from the user's perspective. While this is an arguable position and no doubt some readers will dispute it, at least you know my underlying assumptions. Now, let's see how to access those ROM BIOS functions.

ROM BIOS Calls from Turbo C

Like DOS itself, the ROM BIOS routines under interrupt 10H expect a function code in register AH, with other parameters as required by specific functions in the rest of the registers. Any number of Microsoft, IBM and commercial publications document the ROM BIOS calls; for this project I referred to Ray Duncan's indispensable Advanced MS-DOS (pages 399-420). ROM BIOS calls are made using the Turbo C **int86()** function. Registers are set up in a structured variable bound to the **REGS** union as defined in Turbo C's DOS.H header file.

The **REGS** union includes all the general registers of the 80x86 line of processors and furnishes a notation convention for byte (8-bit) and word (16-bit pair) registers. For example, if you declare the structured variable as **union REGS r;** you can load the value 0CH into byte register AH with **r.h.ah=0x0C;** and the same value into word register BX with **r.x.bx=0x0C;**. The structure notation **.h.** indicates a byte register, and notation **.x.** indicates a word register. The **REGS** union encompasses all byte registers (AH-DL) and all word registers (AX-DX) as well as SI, DI, and the flags. It does not include the segment registers (CS, DS, ES, and SS) which are irrelevant to ROM BIOS calls.

To call a ROM BIOS video service routine, you place the function code in **r.h.ah** and the required parameters in other registers and execute the Turbo C **int86()** function **int86(0x10, &r, &r);**. Function **int86()** performs the following:

* Saves the caller's registers
* Loads the CPU registers from the union whose address is passed in the second argument
* Executes the interrupt given in the first argument
* On return from the ROM BIOS, places register contents in the union passed as the second address argument (the same as the first address argument in this example)
* Restores the caller's registers
* Resumes execution in the calling program

On resumption, you can pluck any BIOS-returned value from the variable bound to the **REGS** union (r in the examples shown here) by using the appropriate register-type notation. For example, suppose you want to determine the current cursor position in video page 0. The setup is:

```
r.h.ah=0x03; /* read position */
r.h.bh=0;        /* in page 0 */
int86(0x10, &r, &r);
                 /* call BIOS */
```

Afterwards you can fetch row with **cursRow=r.h.dh;**. This is the same as, but easier than, performing an equivalent call in assembly language. An added benefit is that the structured variable bound to the **REGS** union retains the returned register values for as long as it exists or until the next call to the ROM BIOS routines. Thus, you can fetch the returned

Figure 1: The Fundamental 16 ROM BIOS Calls

```
/* VIDEO.I: Contains ROM BIOS video calls to be used in Turbo C */

#define  ROM  0x10

union REGS  inreg, outreg;

int videomode (int *ncols)              /* get current display mode */
{                                  /* return number of cols via *ncols */
  inreg.h.ah = 0x0F;
  int86 (ROM, &inreg, &outreg);
  *ncols = outreg.h.ah;                          /* number of cols */
  return (outreg.h.al);                             /* return mode */
} /* ------------------------- */
int activepage (void)                  /* return active display page */
{
  inreg.h.ah = 0x0F;
  int86 (ROM, &inreg, &outreg);
  return (outreg.h.bh);
} /* ------------------------- */
void setmode (int mode)                              /* set video mode */
{
  inreg.h.al = mode;
  inreg.h.ah = 0x00;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void setcursor (int start, int end)             /* set cursor shape */
{
  inreg.h.ch = start;
  inreg.h.cl = end;
  inreg.h.ah = 0x01;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
int curstart (void)                        /* get cursor starting line */
{
  inreg.h.bh = 0;                 /* (cursor shape same in all pages */
  inreg.h.ah = 0x03;
  int86 (ROM, &inreg, &outreg);
  return (outreg.h.ch);
} /* ------------------------- */
int cursend (void)                          /* get cursor ending line */
{
  inreg.h.bh = 0;
  inreg.h.ah = 0x03;
  int86 (ROM, &inreg, &outreg);
  return (outreg.h.cl);
} /* ------------------------- */
void cursoff (void)                                /* turn cursor off */
{
  inreg.h.ch = curstart () | 0x10;                  /* turn on bit 4 */
  inreg.h.cl = cursend ();
  inreg.h.ah =  0x01;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void curson (void)                                 /* turn cursor on */
{
  inreg.h.ch = curstart () & 0x07;     /* turn off high order bits */
  inreg.h.cl = cursend ();
  inreg.h.ah =  0x01;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void gotoxy (int col, int row, int page)          /* set cursor pos */
{                                            /* must specify page */
  inreg.h.bh = page;
  inreg.h.dh = row;
  inreg.h.dl = col;
  inreg.h.ah = 0x02;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
int wherex (int page)                /* return cursor column in page */
{
  inreg.h.bh = page;
  inreg.h.ah = 0x03;
  int86 (ROM, &inreg, &outreg);
  return (outreg.h.dl);
} /* ------------------------- */
int wherey (int page)                  /* return cursor row in page */
{
  inreg.h.bh = page;
  inreg.h.ah = 0x03;
  int86 (ROM, &inreg, &outreg);
  return (outreg.h.dh);
} /* ------------------------- */
void setpage (int page)                 /* set active display page */
{
  inreg.h.al = page;
  inreg.h.ah = 0x05;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void cls (void)                                  /* clear active screen */
{
  inreg.h.al = 25;                                    /* entire screen */
  inreg.h.bh = 0x07;                          /* set to gray on black */
  inreg.h.ah = 0x06;
  inreg.h.cl =  0; inreg.h.ch =  0;
  inreg.h.dl = 79; inreg.h.dh = 24;
  int86 (ROM, &inreg, &outreg);
  gotoxy (0, 0, activepage ());
} /* ------------------------- */
void window (int  x1, int y1,           /* window upper left corner */
             int  x2, int y2,                /* lower right corner */
             char attrib)                 /* text attribute inside */
{
  inreg.h.al = y2 - y1 + 1;                    /* clear entire window */
  inreg.h.bh = attrib;
  inreg.h.cl = x1; inreg.h.ch = y1;
  inreg.h.dl = x2; inreg.h.dh = y2;
  inreg.h.ah = 0x06;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void winScroll (int x1, int y1, int x2, int y2,   /* scroll window */
                int attr)                        /* one line upward */
{
  inreg.h.al = 1;
  inreg.h.cl = x1; inreg.h.ch = y1;
  inreg.h.dl = x2; inreg.h.dh = y2;
  inreg.h.bh = attr;
  inreg.h.ah = 0x06;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
char chattr (int foregrnd, int backgrnd)        /* character attrib*/
{
  return ((backgrnd << 4) + foregrnd);
} /* ------------------------- */
char rdchara (int page,                          /* read char at curs pos */
              char *attrib)          /* return attribute indirectly */
{
  inreg.h.bh = page;
  inreg.h.ah = 0x08;
  int86 (ROM, &inreg, &outreg);
  *attrib = outreg.h.ah;
  return (outreg.h.al);
} /* ------------------------- */
void wrtcha (char ch, char attrib,          /* write char + attrib */
             int page)                  /* at cursor pos on page */
                                  /* NOTE: does not advance cursor */
{
  inreg.h.al = ch;
  inreg.h.bh = page;
  inreg.h.bl = attrib;
  inreg.x.cx = 1;
  inreg.h.ah = 0x09;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void wrtstra (char *str,                              /* write string */
              char attrib,                       /* with attribute */
              int page)                                 /* to page */
{                                  /* starting at cursor position */
int   c, r, n, p = 0;

  videomode (&n);                              /* get width of screen */
  r = wherey (page);                              /* get current row */
  while (str[p]) {
    wrtcha (str[p++], attrib, page);             /* write next char */
    if ((c = wherex (page)) < (n - 1))
      gotoxy (c + 1, r, page);                    /* advance cursor */
    else
      gotoxy (0, ++r, page);                           /* else wrap */
  }
} /* ------------------------- */
void wrtch (char ch, int color,              /* write char in color */
            int page)                 /* at cursor position on page */
{
  inreg.h.al = ch;
  inreg.h.bl = color;
  inreg.h.bh = page;
  inreg.x.cx = 1;
  inreg.h.ah = 0x0A;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void wrtstr  (char *str,                               /* write string */
              char color,                                 /* in color */
              int page)                                    /* to page */
{                                  /* starting at cursor position */
int   c, r, n, p = 0;

  videomode (&n);                              /* get width of screen */
  r = wherey (page);                              /* get current row */
  while (str[p]) {
    wrtcha (str[p++], color, page);              /* write next char */
    if ((c = wherex (page)) < (n - 1))
      gotoxy (c + 1, r, page);                    /* advance cursor */
    else
      gotoxy (0, ++r, page);                           /* else wrap */
  }
} /* ------------------------- */
void palette (int palno)                      /* set color palette */
{                                  /* valid only in mode 4 on CGA */
  inreg.h.bh = 1;
  inreg.h.bl = palno;
  inreg.h.ah = 0x0B;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void graphbackground (int color)          /* set graphics b/g color */
{
  inreg.h.bh = 0;
  inreg.h.bl = color;
  inreg.h.ah = 0x0B;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
void plot (int x, int y, int pixel)           /* plot pixel at x, y */
{
  inreg.h.al = pixel;                        /* pixel (color) value */
  inreg.h.bh = 0;
  inreg.x.cx = x;
  inreg.x.dx = y;
  inreg.h.ah = 0x0C;
  int86 (ROM, &inreg, &outreg);
} /* ------------------------- */
int pixel (int x, int y)                /* return pixel value at x, y */
{
  inreg.x.cx = x;
  inreg.x.dx = y;
  inreg.h.ah = 0x0D;
  int86 (ROM, &inreg, &outreg);
  return (outreg.h.al);
} /* ------------------------- */
```

register values at your convenience. You can translate these ROM BIOS calls into C functions.

Syntactic Sugar

Many of Turbo C's 350 functions and macros are simply DOS and ROM BIOS calls sugar-coated to look like C. In the same spirit, you can pack a little sugar around the ROM BIOS video service routines and call them 'a basic library of Turbo C graphics functions.' Figure 1 shows a #include file, VIDEO.I, which contains the fundamental 16 ROM BIOS calls plus a couple of extended functions that synthesise several calls. Any program that has to perform graphics operations can obtain the full set of functions with the simple statement:

```
#include <video.i>
```

calling whichever specific ones it needs. Because the source code is included in the compilation, the functions will automatically adapt to the memory model currently in use. Also note that every function is

Figure 2: Header to Define Function Prototypes

```
/* VIDEO.H: Prototypes for contents of user-written VIDEO.LIB */
/* Describes calls to ROM BIOS video functions */

int videomode (int *ncols);
int activepage (void);
void setmode (int mode);
void setcursor (int start, int end);
int curstart (void);
int cursend (void);
void cursoff (void);
void curson (void);
void gotoxy (int col, int row, int page);
int wherex (int page);
int wherey (int page);
void setpage (int page);
void cls (void);
void window (int x1, int y1, int x2, int y2, char attrib);
char chattr (int fgrnd, int bgrnd);
char rdchara (int page, char *attr);
void wrtcha (char ch, char attr, int page);
void wrtstra (char *str, char attr, int page);
void wrtch (char ch, int color, int page);
void wrtstr (char *str, char color, int page);
void palette (int palno);
void graphbackground (int color);
void plot (int x, int y, int pixel);
int pixel (int x, int y);
```

Figure 3: The Three routines hdraw, vdraw, and draw

```
/* DRAW.I: Draws lines in graphics mode */

/* hdraw draws horizontal line along y between x1 and x2 */
void hdraw (int x1, int x2, int y, int color)
{
int   x;

  if (x1 > x2) {              /* sort x's into left-to-right order */
    x = x1; x1 = x2; x2 = x;
  }
  for (x = x1; x <= x2; x++)
    plot (x, y, color);
} /* ------------------------ */

/* vdraw draws vertical line along x between y1 and y2 */
void vdraw (int y1, int y2, int x, int color)
{
int    y;

  if (y1 > y2) {              /* sort y's into top-to-bottom order */
    y = y1; y1 = y2; y2 = y;
  }
  for (y = y1; y <= y2; y++)
    plot (x, y, color);
} /* ------------------------ */

/* draw() does lines on the diagonal */
void draw (int x1, int y1, int x2, int y2, int color)
{
double   xstep, ystep, xcum = 0.0, ycum = 0.0;
int      dx, dy;                                        /* deltas */
register x, y;

  dx = x2 - x1;
  dy = y2 - y1;
  if (abs (dx) >= abs (dy)) {                    /* plot along x axis */
    ystep = (double) dy / dx;          /* movement along y axis per x */
    if (dy < 0) {                              /* y travels to the left */
      if (ystep > 0)
        ystep *= -1;              /* adjust for wrong sign from -y/-x */
    } else                                     /* y travels to the right */
      if (ystep < 0)
        ystep *= -1;                                /* adjust as above */
    dx /= abs (dx);                                    /* x increment */
    for (x = x1, y = y1; x != x2; x += dx) {
      plot (x, y, color);
      ycum += ystep;                        /* cum motion along y axis */
      y = y1 + ycum;                                      /* next y */
    }
  } else {                                     /* plot along y axis */
    xstep = (double) dx / dy;          /* movement along x axis per y */
    if (dx < 0) {
      if (xstep > 0)
        xstep *= -1;
    } else
      if (xstep < 0)
        xstep *= -1;
    dy /= abs (dy);                                    /* y increment */
    for (y = y1, x = x1; y != y2; y += dy) {
      plot (x, y, color);
      xcum += xstep;                        /* cum motion along x axis */
      x = x1 + xcum;                                      /* next x */
    }
  }
} /* ------------------------ */
```

defined before being referenced by other functions, thus eliminating any need for a header file containing prototypes (if you program using ANSI C conventions). The down side of using source level #include files such as VIDEO.I is that they waste memory. Turbo C doesn't optimise itself by pulling out unreferenced code. As a result, all the code for all the functions appears in every .EXE program that #includes VIDEO.I, even if the program calls only a single function. My solution to this is to make the function into a linkable library (.LIB) file.

Creating the Libraries

Binding and linking with user-created libraries in Turbo C can be a bit of a chore. In the small model, the inclusion of the VIDEO.I adds 1,376 bytes to the .EXE file; in the large model, 1,520 bytes. The overhead consists of total bytes minus the sizes of the functions you actually call. If that amount of code space is important to you, it might be worth turning VIDEO.I into one or more libraries so that only those routines that are actually used get linked into the .EXE file.

The first step is to write a file VIDEO.H that defines all the graphics functions prototypes. Figure 2 shows this header file, which you should #include in any programs that link with the video library. You'll also use VIDEO.H in creating the library's individual members. Each function file should #include VIDEO.H at the top. Purists might also want to add comments explaining what the function does and to what library it belongs.

When all the functions have been broken out into separate files, compile them one by one with Turbo C to create a matching set of .OBJ files. Now you're ready to make the library.

The .OBJ files produced by Turbo C are in Microsoft-compatible format, so you can use the DOS **lib** utility to combine them into a linkable library. To produce a library called VIDEOS.LIB containing the functions **videomode()** and **activepage()**, for example, type:

```
LIBVIDEOS+VIDEOMODE+
ACTIVEPAGE;
```

Later you can add **setmode()** and **setcursor()** with the similar:

```
LIBVIDEOS+SETMODE+
SETCURSOR;
```

You can of course add more than two modules at a time to the library simply by specifying the library name first, followed by an entire command line of module

The Bugbuster.

"There wouldn't be a SIDEKICK without Atron's hardware-assisted debug tools."*

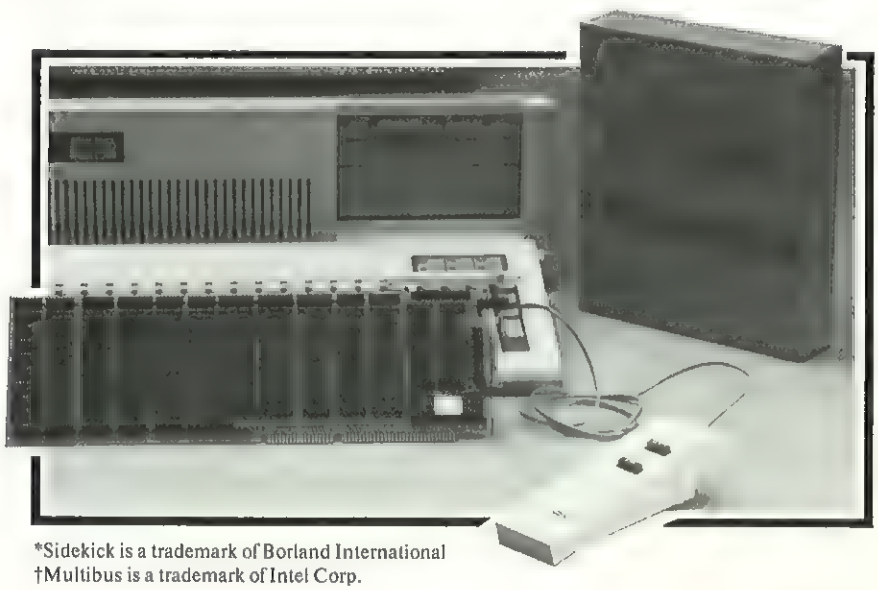
Philippe Kahn, BORLAND INTERNATIONAL

Debugging and tuning your software can take even longer than writing it. The Bugbuster will cut your debug time down to size. Time is more than just money.

Atron's hardware-assisted debug tools are far more powerful than Debug and a fraction of the cost of an in-circuit emulator. Once your code is debugged, the timing analyser will isolate your performance bottlenecks.

Available for the PC, AT and Multibus. The Atron probe is an expansion card that simply plugs into the back of your system. With a link to the CPU's socket, the probe monitors the CPU and I/O channel activity for complex breakpoints and real time trace.

Features; ★ Hardware Breakpoints ★ Real time trace ★ Source level debugging in 'C', Pascal, Fortran or Assembler. ★ Symbolic debugging ★ Performance and timing analysis ★ 80286 Protected Mode support ★ Program crash recovery ★ Full technical support ★ Training from the world's leaders in PC technical training



*Sidekick is a trademark of Borland International
†Multibus is a trademark of Intel Corp.

For further details contact: **QA Ltd.**

Cecily Hill Castle, Cirencester,
Gloucestershire GL7 2EF
Telephone: (0285) 5888 Telex: 437308

CIRCLE NO 795

QA

names separated by plus signs. The S on the end of the library name is a convention borrowed form Turbo C to indicate the memory model; here it is assumed that you compiled the separate units in the small model.

To link with this library, first make sure it's in the directory where Turbo C stores source files (not in the \TURBOC\LIB directory with the vendor's libraries). Put the entry VIDEOS.LIB in your project file (.PRJ). Turbo C then knows what you want to link to a user-created library and where to find it. Repeat the compilation and LIB procedures for each memory model you use, varying the library file name accordingly (VIDEOT.LIB, VIDEOC-.LIB and so on). You'll have to modify the library name in your .PRJ file if you decide to change memory models during the development of a program.

Because it takes an hour or two to get through this whole business of making libraries, weigh the price in time investment vs saving a thousand bytes or so in the .EXE file. It's your choice.

Adapting to the Adaptor

Table 1 describes how to determine which video adaptor is present in your machine. Refer to it for the 'theory' – this article concentrates on the practical aspects of applying the adaptor identification to APA graphics, specifically on the CGA and EGA. The principles discussed here are applicable to other adaptors such as the Hercules as well.

In text modes, there's little adapting to do for a specific display type. The screen is always 25 rows high and either 40 or 80 columns wide. The 40-column mode only applies when writing text on a 320x320 pixel graphics screen; otherwise, you're always in the 80-column mode. The monochrome display adaptor (MDA) can't produce colours but it can do normal, intense, underlined and blinking (attributes remarkably similar to colours). Thus, although the demo program later in this article does a few tricks using the video library, I'm not going to discuss them here. Because text graphics has its own set of problems and solutions, I'll save the discussion for Part 2.

When your software knows the video adaptor it's working with, it can alter its behaviour to suit. After identifying the video board, the software can then adjust its coordinate system or colour selections to fit the APA display. Here, in the interest of limited space, I'll show you how to make a program dynamically alter its coordinate system.

Suppose, for example, that you want to draw a border around the screen starting 15 scan lines (y points) below the top and ending 15 above the bottom. The width in either case is 640 pixels and the top of the rectangle is at y=15. However, the CGA has a mere 200 vertical scan lines, whereas the EGA has 350. Therefore you can set up an assignment condition:

```
if(adaptor == cga)
bottom = 199-15
else
bottom = 349-15;
```

"It takes an hour or two to get through the business of making libraries. Is that worth 1,000 bytes in the .EXE file?"

The trick here is to locate the bottom of the box, which is determined by the adaptor type. The vertical line drawing routine can then repeatedly call the **plot()** function from the video library, with the number of calls – pixels ploted – being determined at run time by the type of adaptor available. Similarly, the horizontal line drawing routine can draw the bottom at the appropriate elevation (y=184 or y=334 for CGA and EGA respectively).

This is a somewhat simplistic scenario implemented in the demo program. A more complex application might consider the CGA default and supply a multiplier of 1.0 in calculating y coordinates. If an EGA is present, however, the program can substitute 1.75 for the y multiplier (because 350/200=1.75). Furthermore, if four-colour graphics are required, it can use 1.0 as the multiplier for the default (mode 04H=CGA 320x200 four colour) and substitute 2.0 if an EGA is present (mode 10H = EGA 640x350 four colour). Thus the program dynamically redefines its coordinate workspace in both dimensions based on the available video adaptor.

A program that does extensive APA graphics will rely on the video library's **plot()** function which plots individual pixels, but it could clearly benefit from some higher-level drawing routines.

Adding APA Graphics

Ultimately, all objects appearing in the display are composed of three kinds of lines: vertical, horizontal and diagonal. Even a solid object such as a block cursor or complex filled polygon consists of some number of contiguous lines arranged so as to approximate a form that the eye recognises. This suggests enhancing the video library with line-drawing routines that you can employ to create shapes.

There's less overhead in drawing orthogonal lines (vertical or horizontal) than in drawing diagonals. Why? Because diagonals proceed along a slope or ratio between vertical and horizontal motion. Slopes are, almost without exception, fractional numbers that entail floating point operations which are inherently slower in computers.

Figure 4: Creating Your Own Colour Identifiers

```
/* COLORS.H.  Maps colour names */

#define BLACK       0       #define DARKGREY       8
#define BLUE        1       #define LIGHTBLUE      9
#define GREEN       2       #define LIGHTGREEN    10
#define CYAN        3       #define LIGHTCYAN     11
#define RED         4       #define LIGHTRED      12
#define MAGENTA     5       #define LIGHTMAGENTA  13
#define BROWN       6       #define YELLOW        14
#define LIGHTGREY   7       #define WHITE         15
```

Program Development Tools from Roundhill

PANEL Plus

Advanced Screen Manager

PANEL Plus is a screen management library for the C language, supplied with full source code. Utilities are included to design screens interactively and to generate C code for screen data areas and control blocks. Programs and screen layouts are portable to all supported compilers and operating systems. MS-DOS versions £325.00. Also supports OS/2, Xenix, Unix, and VMS.

M.E. Programmers' Text Editor

M.E. (Macro Editor) is a full-featured programmers' text editor, with multiple windows, column cut and paste, a C-like macro language, and full source code. The editor supports regular expressions, keyboard macros, wordwrapping, brace matching, auto-indentation, and multiple scrap buffers. Licences are available to incorporate M.E. into your own products. £75.00.

Periscope Debugger

Four models of hardware-assisted, symbolic, source-level debugger, ranging in price from £115 to £695. Debugging code remains resident and a breakout switch allows a running program to be stopped at any time to investigate problems. The Periscope III version is a full-hardware debugger which monitors the system bus and allows a program to be trapped on specific address accesses.

Lattice

Amiga C Compiler V4.00

The latest version of Lattice C for Commodore Amiga. Features improved compilation speed, direct calling of Amiga functions, full ANSI support, overlay linker, and assembler. £125. Upgrades from earlier Lattice C versions are available from Roundhill.

Polytron Version Control System

The leading source-code management system for MS-DOS and VMS. Corporate version for MS-DOS £250 (for Polymake make utility, add £95). Network versions are available from £625. PVCS allows multiple versions of text to be retained and keeps a complete revision history. Separate lines of development can be created using "branching", and simultaneous changes can be merged.

Roundhill Computer Systems

Roundhill Computer Systems Limited, Axholme, London Road, Marlborough, Wiltshire SN8 1LR Tel: (0672) 54675
(The prices shown exclude VAT and are subject to exchange rate adjustment)

CIRCLE NO 796

Figure 5a: Graphics Demonstration Program (Part 1)

```
/* VID.C: Demos video functions */

/* TURBO C INCLUDES */
#include <stdio.h>
#include <bios.h>
#include <dos.h>

/* USER-WRITTEN INCLUDES */
#include <colors.h>
#include <video.i>
#include <draw.i>

/* LOCAL FUNCTION PROTOTYPES */
int  vidIdent (int *vidmode);
void wait (void);
void stairsteps (void);
int  isEGA (void);
void label (void);
void bigX (int adap, int vmode);
void hourglass (int adap, int vmode);

/* GLOBALS */
enum vidTypes (mda, cga, ega, compaq, other);
main ()
{
int  adaptor, mode, cols;

  cls ();                                        /* clear screen */
  adaptor = vidIdent (&mode);            /* identify video adaptor */
  if (adaptor != other) {
    stairsteps ();                      /* cursor positioning demo */
    bigX (adaptor, mode);                     /* graphics demo #1 */
    hourglass (adaptor, mode);                /* graphics demo #2 */
    label ();
    gotoxy (36, 12, 0);
    puts ("All done!");
  }
} /* ------------------------ */
int  vidIdent (int *vidmode)             /* identify video adaptor */
{
int  flag, adap, width;

  label ();                                        /* label display */
  puts ("\n\nDISPLAY INFORMATION:");
  *vidmode = videomode (&width);                  /* get video mode */
  flag = (biosequip () & 0x18) >> 4;          /* get video eqpt flag */
  if (isEGA ()) {
    adap = ega;
    puts ("\n\n  Enhanced Graphics Adaptor");
  } else
    switch (flag) {
      case 0: if (*vidmode == 2) {
                adap = compaq;
                puts ("\n\n  Compaq adaptor");
              } else {
                adap = cga;
                puts ("\n\n  Color Graphics Adaptor");
              }
              break;
      case 3: adap = mda;
              puts ("\n\n  Monochrome Display Adaptor");
              break;
      default: adap = other;
              puts ("\n\n  Adaptor not usable in this demo. Sorry.");
    } /* end of switch */
  printf ("\n  Text screen size is %d columns x 25 rows\n", width);
  if ((*vidmode < 4) || (*vidmode == 7))
    puts ("\n  Text mode currently active");
  else
    puts ("\n  Graphics mode currently active");
  wait ();
  return (adap);
} /* ------------------------ */
int  isEGA (void)                      /* determine if EGA is attached */
{                                    /* return TRUE if so, FALSE if not */
  return (peekb (0x40, 0x87));                    /* check EGA info byte */
} /* ------------------------ */

void wait (void)           /* prompt to continue, wait for keypress */
{
int   tab, width;

  videomode (&width);                          /* get width in columns */
  tab = (width - 33) / 2;                  /* starting column for text */
  gotoxy (tab, 24, 0);
  wrtstr ("Press any key to continue demo...", WHITE, 0);
  getch ();
  cls ();
} /* ------------------------ */
```

Slope is not an issue when drawing a line that is perfectly vertical or horizontal, and thus it can be done with more efficient integer arithmetic.

For efficiency, I developed orthogonal line routines and a separate, less efficient routine for diagonals. The latter is called if I know or suspect that the line is not orthogonal.

In the orthogonal routines, you pass the start and end points, the axis coordinate along which the line is to proceed, and the colour of the pixel you want plotted. The routine then uses integer arithmetic to control a loop that plots the individual points. Figure 3 supplies two such routines, called **hdraw()** and **vdraw()**, for horizontal and vertical lines respectively.

Because callers will probably use these routines frequently, passing variables as arguments, you should not place on them the burden of ordering the start and end points to suit the expectations of the routines. Consequently, these routines sort the start and end into the low- and high-order they expect. The calls **hdraw (35, 231, 20, 1);** and **hdraw(231, 35, 20, 1);** both produce exactly the same result; namely, a horizontal line between x coordinates 35 and 231 inclusive along y coordinate 20 in colour index 1.

In considering the diagonal routine **draw()**, it's important to recognise several factors. The requested line might be orthogonal, resulting in a slope that is either 0 (horizontal) or infinity (vertical). To avoid division-by-zero failures, the routine must anticipate these problems in advance and take corrective action by preassigning the correct step value.

The diagonal must move along the axis that results in the densest line, or in other words, along the axis that has the greatest distance to travel. If the motion is greater along the x axis than along the y, the more dense line results from plotting each y intersection per x increment. Thus the routine must determine the axis with the greatest movement and make that the controlling axis.

The stepping value along the short motion axis is a floating point number that is the ratio of short-motion to long-motion. For example, if the x axis moves +120 points and the y axis -30, the x axis is controlling and the y moves -0.25 points per x (-30/120=-0.25), which is its slope relative to the controlling axis.

The stepping value is additive. That is, for each controlling increment, the stepping value is added to the current controlled value. This motion accumulates and is

Figure 5b: Graphics Demonstration Program (Part 2)

```
void label (void)                    /* label the screen at top center */
{
  gotoxy (30, 0, 0);
  wrtstra ("Video demonstration", chattr (YELLOW, BLUE), 0);
} /* ------------------------ */
void stairsteps (void)
{
int   color = 1;

   label ();
   gotoxy (31,  2, 0); wrtstr ("Cursor positioning", LIGHTGRAY, 0);
   gotoxy (10,  4, 0); wrtstr ("Stair", color++, 0);
   gotoxy (20, 10, 0); wrtstr ("steps", color++, 0);
   gotoxy (30, 16, 0); wrtstr ("going", color++, 0);
   gotoxy (40, 22, 0); wrtstr ("down",  color++, 0);
   gotoxy (50, 16, 0); wrtstr ("and",   color++, 0);
   gotoxy (60, 10, 0); wrtstr ("back",  color++, 0);
   gotoxy (70,  4, 0); wrtstr ("up",    color, 0);
   wait ();
} /* ------------------------ */
void bigX (int vidAdap, int vmode)          /* APA graphics demo #1 */
{                       /* draws full-screen border and X, adjusting */
                                    /* for EGA or CGA as indicated */
int  x1 =  0, x2 = 639;
int  y1 = 15, y2;

  if ((vidAdap == mda) || (vidAdap == other))        /* can't do it */
    return;                                  /* so return with no action */

  if (vidAdap == ega) {             /* EGA demo: 640 x 350 (mode 0Fh) */
    y2 = 349 - 15;                   /* set bottom of graphics screen */
    setmode (0x0F);                  /* go to EGA mono graphics mode */
  } else {                  /* CGA|Compaq demo: 640 x 200 (mode 06h) */
    y2 = 199 - 15;
    setmode (0x06);
  }
  label ();                                          /* label the screen */
  hdraw (x1, x2, y1, 1);                        /* draw line across top */
  hdraw (x1, x2, y2, 1);                               /* then bottom */
  vdraw (y1, y2, x1, 1);                             /* down left side */
  vdraw (y1, y2, x2, 1);                            /* down right side */
  draw (x1, y1, x2, y2, 1);                          /* main diagonal */
  draw (x1, y2, x2, y1, 1);                          /* cross diagonal */
  wait ();
  setmode (vmode);
} /* ------------------------ */
void hourglass (int vidAdap, int vmode)          /* graphics demo #2 */
{                   /* operates in 320 x 200 four-color (CGA) mode */
int     y, x1 = 60, x2 = 260, pixval = 1;

  if ((vidAdap == mda) || (vidAdap == other))        /* can't do it */
    return;                                          /* so go back */
  setmode (4);                                /* go to CGA graphics mode */
  gotoxy (8, 0, 0);
  puts ("320 x 200 color graphics");                    /* show mode */
  palette (0);
  for (y = 50; y < 151; y++ ) {                        /* draw figure */
    hdraw (x1, x2, y, pixval);
    x1 += 2; x2 -= 2;                                   /* change x's */
    if (y ==  84) pixval = 2;                         /* change colors */
    if (y == 117) pixval = 3;
  }
  wait ();
  setmode (vmode);
} /* ------------------------ */
```

truncated or rounded for each controlling step, according to the way the routine is written (Figure 3 truncates it). These considerations enable the routine to plot a diagonal or orthogonal line in any direction. The cost of this flexibility is slower performance.

As you use the video library, you'll undoubtedly add your own graphics routines for such things as polylines, filled shapes, circles, arcs and so on. The DRAW.I file given here is merely a suggestion of the kinds of things you can do to make graphics easier in Turbo C.

You can also make life a little easier, and your source code more readable, by including the file COLOURS.H (see Figure 4) in programs that require colours. It's always easier to understand identifiers than magic numbers requiring you to memorise, for example, that 07H is light grey.

A Graphic Sampler

Now let's put this discussion to work in a demonstration program. Figure 5 contains the demonstration program VID.C, which calls a number of the routines in the video library. It produces some simple but illustrative effects in both text and graphics modes.

The program first identifies the adaptor on the host machine so that it can subsequently tailor its behaviour to suit (or bypass operations that the adaptor can't handle such as APA graphics on an MDA). This program is written to run on the MDA, CGA, EGA and Compaq; it doesn't recognise the Hercules card.

Next, it provides a text graphics display that showcases cursor positioning with **gotoxy()** and (if the monitor is capable) colour text. Except for the string-writing functions that produce a label at the top and a prompt at the bottom of each demonstration display, this is the only use of text graphics in the program.

In the third display, the program draws a border around the screen and a large corner-to-corner X inside it. It calls the functions in DRAW.I. This routine shows how to make a run-time adjustment to the coordinate system according to whether the adaptor is a CGA (or Compaq) or an EGA. If you're running with an MDA, the program bypasses this display and the next because the hardware can't handle it.

The fourth panel is in CGA 320x200 pixel four-colour graphics. It draws a three-colour hourglass illustrating two things: first, that solids on a display are indeed composed of numerous horizontal lines and second, that the EGA (if that's what you're using) can emulate a CGA monitor.

The final display merely announces that the demo is finished. If you haven't converted the video functions into a linkable library but instead are using the code in Figure 1 directly, change the indicated directive to #include<VIDEO.I>near the top of the file before compiling. On the other hand, if you've gone to the trouble of creating VIDEO.LIB, set up a project file VID.PRJ containing the following entries:

```
vid
video.lib
```

Make sure VIDEO.LIB is in the same directory as your source program VID.C, then start the compile, link and go session with Alt-R.

Kent Porter is a prolific programming author based in the US, with 17 books and hundreds of magazine articles in print. He can be contacted on 1909-4 Montecito Road, Mountain View, CA 94043, US and is a technical editor for Dr. Dobb's Journal. *This article first appeared in the November 1987 issue of Dr. Dobb's Journal of Software Tools, published by M&T Publishing Inc, 501 Galveston Dr, Redwood City, CA 94063. It is reprinted with their permission.*

Identifying Video Adaptors on the IBM PC

ROM BIOS interrupt 10H on IBM PCs and compatibles furnishes video services to accommodate a variety of hardware configurations. Although many of the calls - notably those dealing with text - are device independent, others are device specific. If your graphics software is to function effectively in this kind of variable environment, it must adapt its behaviour to suit the hardware. And to do that, it must first find out what the video hardware is.

This isn't as easy as it might sound. Programs that want to determine the video capabilities at their disposal usually have to look in several places piecing together many clues.

First, I'll discuss sources of information about the video environment, then I'll show how to use them to identify the video hardware. At memory address 40H:07H, the ROM BIOS keeps a byte called the equipment flag which describes the machine's hardware configuration. Bits 4 and 5 give information about the attached video device. By masking out the other bits and shifting right four places, you can isolate a device number from 0 to 3. In C, for example, you might write this operation as:

```
adaptor=(peek(0x40,0x07) &0x18)>>4;
```

A similar effect can be obtained in BASIC with the statements:

```
DEF SEG &H40
ADAPTOR = (PEEK (7) \\ 8) AND &H03
```

The full range of resulting values of the BIOS video equipment flag is:

0	undefined (used by Compaq, EGA and CGA)
1	40x25 B/W text, colour graphics (also used for non-IBM video devices such as Hercules)
2	80x25 text, CGA (This is unreliable).
3	80x25 text monochrome.

As can be seen, the BIOS video equipment flag raises more questions than it answers. The only thing that's sure is that when its value is 3, you have a monochrome (that is, text only) adaptor. Even so, you might want to check further; if the video mode discussed next is 7, it's definite that the video device is incapable of APA graphics and colour text.

The Video Mode

The ROM BIOS furnishes two methods for obtaining the video mode - that is, the current mode of operation for the adaptor. One is to call interrupt 10H with function code 0FH in the AX register. On return, AL contains the mode value. An alternative is to bypass the ROM BIOS and do yourself what function 0FH does indirectly: read the video mode byte from location 40H:49H.

Whichever, the result is a byte whose possible settings are shown below:

Mode	Resolution	Description
00H	40x25	B/W text, colour graphics (obsolete)
01H	40x25	colour text
02H	80x25	B/W text
03H	80x25	colour text
04H	320x200	4-colour graphics
05H	320x200	4-colour graphics (colour burst off)
06H	640x200	2-colour graphics
07H		monochrome adaptor or EGA text mode
08H	160x200	16-colour graphics (PCjr)
09H	320x200	16-colour graphics (PCjr)
0AH	640x200	4-colour graphics (PCjr)
0BH		not used
0CH		not used
0DH	320x200	16-colour graphics (EGA)
0EH	640x200	16-colour graphics (EGA)
0FH	640x350	2-colour graphics (EGA)
10H	640x350	4-colour or 16-colour graphics (EGA) depends on available display memory.

These values can also be used to set the video mode, assuming the attached device is capable of supporting them. The values suggest the capabilities of the monitor by inference only and can be misleading. For example, although mode 07H indicates an active monochrome display adaptor (MDA), it is also valid for the EGA in plain text mode.

Note that modes 0BH and 0CH are undefined. Presumably, these are reserved for future developments or for graphics adaptors that were never introduced.

The EGA

If the current mode is anything less than 08H, you have to investigate further to find out if an EGA card is present in the system. The ROM BIOS furnishes no call for this, so you have to read memory location 40H:87H to find out. This address contains the EGA information byte.

Although each bit or bit pair in the EGA information byte has some unique significance, in general it's sufficient to know that the byte is 0 when an EGA is not present and nonzero when one is. Therefore, in C, you can define a Boolean function **isEGA()** as follows:

```
char isEGA(void)
{
  return (peekb (0x40, 0x87));
}
```

which returns FALSE when there is no EGA attached and TRUE (some non zero) when one is. Similarly, you can test for an EGA monitor in BASIC with statements such as:

```
DEF SEG &H40
IF PEEK (&H87) <> 0 THEN (EGA
present) ELSE (not)
```

A brief discussion of the EGA is perhaps a worthwhile digression here. The EGA furnishes several extended APA graphics capabilities. That is, you can have more pixels if higher density with more colours than in the earlier graphics adaptors. How the EGA works is beyond the scope of this article. What's important is that the EGA is downward compatible. That is, it supports all the modes available in the earlier MDA and CGA plus some of its own. The EGA offers no text capabilities beyond the CGA, except that the characters are more dense and thus more readable; this is a given and beyond programmer control. Otherwise, the same combinations of 16 foreground by 16 background colours are available.

The advantages of the EGA over other adaptors thus involve only APA graphics. In text modes, the only discriminator is whether the display is pure monochrome (MDA) or not. If it's pure MDA, you can't do colours, otherwise you can.

Other Common Adaptors

The popular Hercules card offers APA graphics enhancements but nothing extraordinary by way of text. The Hercules provides for grey shading of text as a substitute for text colours and APA graphics on a 720x348 pixel monochrome display.

The Compaq adaptor is second only to the EGA in versatility. In text mode, it acts like the EGA's text mode, producing high-resolution text in any combination of 16 foreground and 16 background colours. You can also switch it into any of the CGA graphics modes (04H-06H) and it behaves just like the CGA adaptor. Identifying a 'Herc' or a Compaq, like the others, is a matter of recognising a pattern of indicators.

The Software Sherlock Holmes

Now let's see how these clues fit together. The signatures of several popular video adaptors are shown in their default (power-up) conditions:

	EGA	CGA	MDA	Com.	Herc
BIOS Equipment	0	0	3	0	1
Video Mode	3	3	7	2	7
EGA Byte	non-0	0	0	0	0

Notice that each adaptor has a unique pattern of values. Given this, you can quickly sift through all three items of information and identify the adaptor. Expressed in pseudo code, the algorithm is:

```
if EGA byte <> 0 then
  adaptor = EGA;
else

  case BIOS equipment flag of:
  0:  if video mode=2 then
        adaptor = Compaq
    else
    adaptor=CGA;
    endif;
    break;

  1:
    adaptor=Hercules;
    break;

  3:
    adaptor=MDA;

  endcase;
endif;
```

Note: some other monitors use the same signature as the Hercules does. An example is the MDS Genius full page display, popular with desktop publishing systems. In this case, you have to look at the display buffer size which you can fetch as an integer from 40H:4CH. The Hercules display buffer is 16,384 bytes: that for the Genius is 8,192 bytes.

Obviously then, the table above and the algorithm are not comprehensive. The great majority of video adaptors for IBM PCs and compatibles emulate one of these de facto standards however, and thus the method here is reliable in almost all cases.

RPC: The Key To Distributed Software

The term 'Network Computing' is now second nature to workstation manufacturers. Its accommodation of 'distributed applications' hinges on the use of Remote Procedure Calls. Herrick Johnson and Mark Adams delve into Apollo's RPC.

Assume for a moment that the quagmire of networking standards has been miraculously resolved. Assume that networks are taken for granted, conflicting protocols absent and connectivity problems a thing of the past. With all this in place, as a software developer, you've suddenly got an awesome hardware power sitting beneath any program you write. This is what the industry is now calling 'Network Computing'.

Just as an operating system will automatically schedule tasks, allocate priorities, and select appropriate hardware on one machine, so Network Computing allows the same across different machines on a network. If your program ran on a workstation, it could conceivably command the resources of a Cray for one part of the program or a dedicated graphics machine for another. At its ultimate, perhaps large programs could be split across any workstation which had spare CPU cycles! Super-computers could be used in the background, with a windowing system on a PC providing a familiar user interface. And the same application should run on a network of two computers or a network of two-hundred computers in a similar fashion. Even in a pretty humble configuration of PCs, you could download programs to compile on a spare machine or split the compilation task among different machines.

The difference between workable reality and wishful thinking is how the division of processing between machines happens. Particularly:

How much software needs to be rewritten.

How will performance degrade over networks.

The systems vendors of the mid '80s, however, have already spent many years working with this concept on the basis that it would become a working reality. Indeed, significant progress has been made in achieving 'seamless' connection of different kinds of hardware and to an extent this reality now exists. Reason enough, we believe, for software developers now to take Network Computing very seriously indeed.

The Software Utopia

Some of the most popular and complex software of our age is CAD/CAM and CAE (Computer Aided Engineering) software. It uses almost every inch of graphics, storage and processing power available. Software developers are handicapped by the power, resolution and storage their hardware can now deliver. The sheer heat of the MIPS war, and the race to develop and market RISC chips shows just how hungry software users are for more power.

Table 1: RPC - The Key to 'Dispersed' Applications

The *Network Computing Architecture* builds on *Workstation Architecture* which provides a single file system and single security system across a group of computers. Each CPU advertises the availability of unused resources that dispersed applications can then use. *Remote Procedure Calls* are among the tools, information brokers, locating brokers and servers provided by the Network Computing System from Apollo which make dispersed applications possible. Application developers will find the components of the system extensible to solve new problems and designed to be open. Applications will thus evolve to take advantage of new specialised hardware platforms and large numbers of general purpose computers.

A remote procedure call, or RPC, looks like a local procedure call, but allows a client process on one machine access to data, software or a device on another machine that was previously only accessible to a process running on that second machine. An RPC causes a server process running on the target machine to execute some software subroutines on the client's behalf. The RPC is the fundamental tool for building dispersed applications which are applications that are divided into pieces that execute on a variety of systems. To take advantage of multiple CPUs, an application needs to be broken down into parts that are independent of each other. RPCs take a single application that is divided into procedures and provides the framework for some of these procedures to execute on other CPUs.

RPC calls are network independent, and use the Berkeley UNIX 'sockets' system to provide communication across the network. (For a full explanation of sockets see the article entitled 'UNIX-hosted Real-Time Development' in this issue of .EXE).

Herrick Johnson

Figure 1: The Client-Server Model and the role of application, stub and RPC runtime

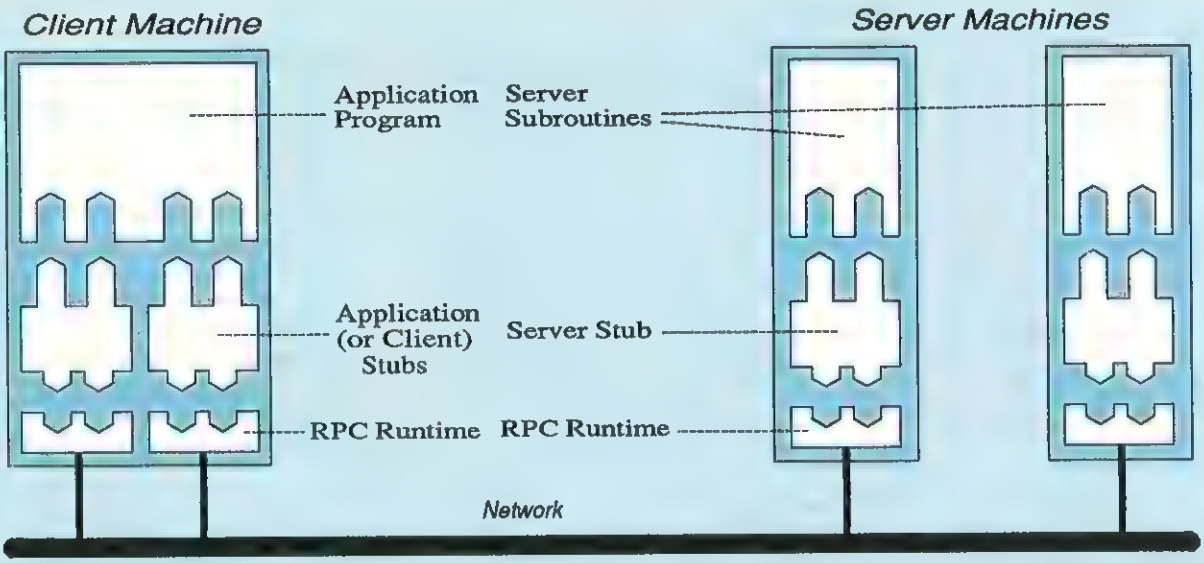
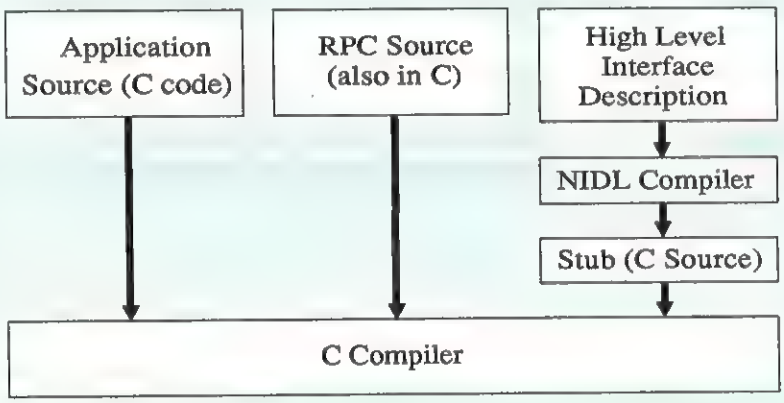


Figure 2: Software Relationships

RPC is written in C and source is supplied. The NIDL compiler produces the stub in C. The RPC, stub and application (also in C) and combined at compile time.



A look at the elements of a typical design package shows just what is demanded. The whole application will centre around a database of some sort. This may incorporate any number of sophistications. Rule checking will be added. Capture, display, testing, simulation and analysis will all typically be performed as well. The demands this makes on processing power mean that very often verification or simulation will be passed off to a batch process. This reduction in interactivity directly affects the productivity of the user of the software. A Network Computing environment could allocate the different processes to different machines on the network and restore interactivity to the user.

The Broker and RPC

There are two very fundamental new concepts to make Network Computing feasible. They both hinge around the vital need to make software modules completely portable. Provided that no specific information about the hardware has been hard-coded into an application, any program should be automatically distributable across a network. The first concerns how a program knows what other hardware and software is on the network. Generally, a specific program will be running to answer this. Sun uses the term 'Yellow Pages', Xerox uses 'Clearing House' and Apollo uses a 'broker' program. The second is the way programs communicate with each other and with different hardware over the network. Here, the Remote Procedure Call is now a popular route.

The broker is best described with the Apollo Client/ Broker /Server model. A client is an end-user or a running program which makes requests on a server. A server could be another running program or a separate computer. Computers as database servers and programs such as windows or graphics servers are now common concepts and the client/server model is common enough now in the UNIX environment. The Broker, or more specifically, the Location Broker, is introduced to keep information about the network and translate user-relevant names and attributes into network IDs so servers can be easily found and used. Servers register their capabilities with the location broker at runtime to determine which hosts to use for particular elements of the software. Clients enquire about current services from the location broker which provides a dynamic way to find and use new and changing resources. The concept of a broker is crucial to the administration of a Network Computer Architecture.

The location broker uses an object oriented approach to network computing. RPC calls are treated as operations on objects, not on particular machines or server processes. By approaching the problem from the standpoint of what to do rather than where to do it, applications can function even in a constantly changing environment.

The Remote Procedure Call (RPC) is a program procedure call which is able to work across a network and is a fundamental tool for building network applications. It handles the packaging, transmission and reception of data and instructions between server and client

Table 2: RPC System Calls

Client Calls	
rpc_$bind	Allocate a handle and create an association between the client and the object's location.
rpc_$free_handle	Release the handle and eliminate the client-object association.
lb_$lookup_interface	Find the network addresses of all the objects that support a particular interface.
lb_$lookup_object	Find the network addresses of all the instances of a particular object.
lb_$lookup_type	Find the network addresses of all the objects of a particular type.
Server Calls	
rpc_$use_family	Add a network protocol family to the list of protocols used by this server to accept RPC calls.
rpc_$register	Register an interface with the RPC runtime library.
rpc_$listen	Listen for RPC calls from clients.
lb_$register	Register an object and an interface with the location broker so that clients in the network can make RPC calls on that interface.
lb_$unregister	Unregister an object and an interface.

subroutines. In practice, using RPCs without any special development or runtime tools is likely to be more hassle for software developers and hence a major hindrance to the viability of CNA. Some clever cosmetics are needed to make RPC work in practice (this is discussed later).

Using RPC

An applications developer will use RPC having written the main program and all subroutines required to run on server machines across the network. These subroutines can, and should, be tested on the machines on which they are finally to run to ensure proper operation (see Figure 1).

The RPC runtime source code must then be able to run on all the machines on which these remote procedures are to run. The source code is made available for adaptation if necessary. These RPC runtime modules provide a smooth and consistent interface across the network. Error handling is part of the RPC runtime, and so RPC does not need to rely on the network's own form of error handling. As a consequence, it can run on any connection, network or protocol that provides point-to-point datagram services. These include UDP/IP, ISO, CLNS, MAP/TOP, IDP in Xerox' NS and Apollo DDS.

Different network protocols and various operating systems are compensated by a user-mode subroutine library accessed by the RPC runtime. This library basically provides a simple extension to the UNIX 'sockets' approach to inter-process communications.

The error handling part of RPC compensates for lost, duplicated and long-delayed messages, messages arriving out of order and server crashes. The runtime also ensures that no call is ever executed more than once. And because the error handling built in to the RPC runtime, the client application can call for only as much error correction as is needed. For example, if a subroutine can be executed more than once without side effect, the overhead to guard against this can be eliminated.

Finally, the RPC runtime environment includes data conversion routines. Source code for routines for changing byte order and converting floating point representations, for example, is provided and is used to create a runtime procedure that handles data conversion between any two types of machine.

The clever stuff comes when linking the remote subroutines to the RPC runtime. This is done through the use of a Network Interface Definition Compiler.

The NIDL

The Network Interface Definition Language Compiler (NIDL Compiler) allows the programmer to specify in a high level form, the necessary code to link the RPC runtime to the application routines. The small amount of code needed to do this is called a **stub** procedure and is generated by the creation of an **interface definition**. The interface definition simply combines location information for all elements of the client and server software and lists the numbers and types of arguments to enable the server routine to run as normal.

The whole RPC is written in C and the NIDL Compiler also produces C source code as its output. This is compiled on the target machine. In fact, the NIDL Compiler generates source for both the client and the server, and produces the necessary #include files.

The NIDL Compiler support three types of binding to link procedures across machines on the network. Binding is done with bind calls that are either explicit, implicit or automatic. In explicit binding, the NIDL states exactly which host to use and this host is always used when the application is run. In implicit binding, the client establishes the binding as a variable before making any remote procedure calls. Thus, the application can query the location broker at runtime to establish the binding between the local and remote routines. Automatic binding is used when a procedure cannot be attached specifically to one machine. Examples are procedures that need to access different hosts at different times, such as such as a routine that gets data from a remote site on the network which needs to bind to a different host depending on where it wants to get its data. Each time the procedure is invoked, the stub makes a call to find the network address of the object to be accessed and then makes a bind call to create the proper binding.

An example of NIDL code serves to illustrate the point best. Suppose a prog ram references a subroutine called **vector-$add** on a vector processor elsewhere on the network. Arguments to the subroutine are two integers, **int a** and **int b** and the one returned argument is the addition of the two, **int c**. The programmer writes the interface routine in the NIDL and starts by giving the interface a name (such as **vector**) and specifying a unique identifier for the interface (this can be generated automatically by a special routine). The location of the #include file

that defines the datatypes used by the vector interface is given using a compiler keyword called **import**. Then, the input and output variables are specified using the keywords **in** and **out**. A system call, **rpc-$handle-t** is also used to give the handle to link the client and server: this is included using the **binding** keyword. Finally, other routines which use the same interface are specified. The resulting code ready for compilation by the NIDL compiler would look like this:

```
[uid (0x1775923a, 0x9c0001d22)]

interface vector {

import"/usr/include/idl/rpc.idl";
  void vector_$add(
    [binding] rpc_$handle_th,
    [in]    int a[],
    [in]    int b[],
    [out]   int c[]
    );

  int vector_$sum (....);
  void vector_$subtract(....);
}
```

Conclusion

RPC provides a means of creating network-independent distributed software, and to an extent epitomises the software developer's utopia of software portability with maximised performance. According to Apollo, NFS and Xerox's XNS versions of RPC require TCP/IP and XNS, making them non-independent, though TCP/IP is now extremely well supported by most manufacturers. The Network Interface Definition Language provides a means of specifying stubs which NFS RPC doesn't and which XNS provides through a Courier language. ID Language NFS. Apollo's attention to the fact that networks are rarely static entities, but are always being reconfigured shows in their use of binding and locating network services. Unlike Sun's Yellow Pages, Apollo's RPC is a dynamic way of locating network services and the choice of three binding options (see above) gives two extra ways of binding over Sun's and Xerox's explicit binding.

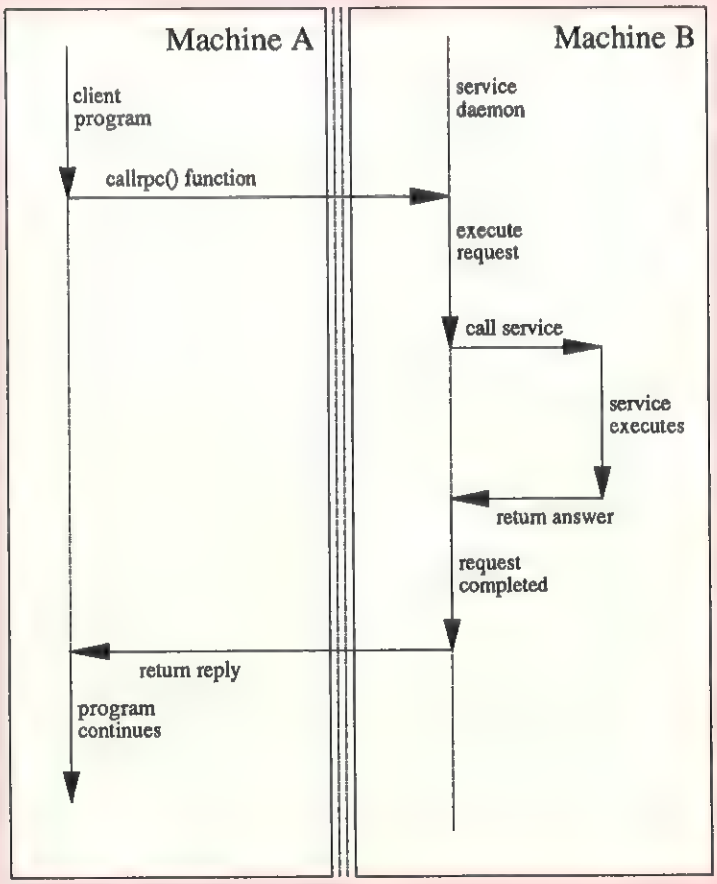
The use of RPC, on whatever workstation, should now be actively considered by software developers as a major step into network programming.

Herrick Johnson is Apollo Computer's Distributed Systems Product Manager and details in this article are taken from the Apollo white paper entitled 'Network Computing Architecture: An Overview'. Mark Adams is .EXE's Surveys Editor. Apollo reference documentation includes 'Network Computing Architecture: Protocol Specifications', 'Network Computing System Reference' and 'Concurrent Programming Support (CPS) Reference'.

Table: Sun Microsystems and RPC.

In March 1984, Sun Microsystems had their Remote Procedure Call libraries running on a 68010-based Sun-2. The first kernel-to-kernel RPC calls were made in June that year. It originally ran using the DARPA User Datagram Protocol (UDP) and the Internet Protocol (IP). Users at that time who worked with networking systems had just two choices for an RPC implementation: Sun's NFS (which incorporated RPC) and Xerox's Courier protocol. Sun has made great market gains with NFS, due in no small part to the fact that source code was readily supplied to customers - a route taken by Apollo since. RPC and a communications protocol called XDR evolved to become the core protocol system for Sun's NFS.

Sun's RPC interface is divided into three layers. The highest layer is totally transparent to the programmer - a call to return the number of users on a remote machine, **rnusers()**, for example, would be used exactly as a local **malloc()**. At the middle layer, the routines **registerrpc()** and **callrpc()** are used to make RPC calls: **registerrpc()** obtains a unique system-wide number, while **callrpc()** executes a remote procedure call. **rnusers()** is implemented using these two routines. The middle layer routines are designed for most common applications, shielding the user from knowing about sockets. The lowest layer is for more sophisticated applications such as altering the defaults of the routines. At this layer, you can explicitly manipulate sockets that transmit RPC messages. Sun advises against using this level!



SQL Part 1: Setting the Database Scene

Russell Jones launches headlong into a six-part series which looks at the role and use of SQL – the database query language.

This is the first in a series looking at the Structured Query Language (SQL) for relational databases. In the series, we're going to look, amongst other things, at the history and development of SQL, at its current level of functionality, at its current problems, at the way it is implemented within particular products, and its role in helping to build 'star' databases of interconnecting computer systems.

In order to understand why SQL has become so important, we must first look at the development of relational database technology over the past fifteen or so years, and at why the technology is proving so attractive to users and software developers alike. This introductory article places SQL within the wider, relational context.

The Evolution Of Relational Databases

The relational database story started in 1970 when Ted Codd, then an IBM researcher, published a now classic paper, 'A Relational Model of Data for Large Shared Data Banks'. In this paper, Codd laid down a set of abstract principles for database management – the so-called relational model. The entire field of relational database technology has its origins in that paper.

One of the first organisations to build upon Codd's work in the 1970s was IBM itself, with its System R Project. One of the products to emerge from the project was SQL. Thanks in large part to the success of the System R Project, other vendors also began to construct their own relational databases. In fact, at least one such product, Oracle, was actually introduced to the market prior to IBM's own products. The PC software market blossomed at just about the time that the whole concept of relational technology was taking off. That's the major reason why all the major PC databases, such as dBase II and III and Rbase, are, to a lesser or greater extent, based around the relational model.

Characteristics of an RD

What features distinguish a relational database? According to Ted Cod, who has since become the recognised relational database guru – a database can be called relational if:

* It stores data in a tabular form following some kind of standard data structure, such as third normal form or the entity relationship model.

* It presents data to users and programmers as a collection of tables – or, to give them their exact name, relations.

Table 1: The Relational Approach to Database

A relational database is one in which the data it contains is perceived by the user and the software developer as a collection of tables. This tabular view of data is easy to understand - we come in contact with tables of data every day in telephone directories, train timetables, business reports, etc.

The user sees his data in two-dimensional tables - building blocks - consisting of rows and columns of data. Figure 1 shows our sample 'company' database overleaf. It contains **EMPNUM, NAME, DEPT**, and **SALARY**. Each employee has a row of information in the table. Tables in a relational database are related dynamically through the values of the information found within them - as in **DEPT** and **DEPTNO** in our example. There are no inbuilt physical links. Indeed, part of the power and flexibility of a relational database is this ability to relate information dynamically, and not to have to have it predefined into the data structure.

In our example, suppose we need to know the name of the department in which 'Smith' works. We have available the **EMPLOYEE** table and the **DEPARTMENT** table. First, in the **EMPLOYEE** table, the **DEPARTMENT** in which 'Smith' works must be identified as 'D12'. This is then matched with the **DEPTNO** in the **DEPARTMENT** table, thus providing the **DEPTNAME** of 'Sales'.

It is the operators of SQL that which provide this function of 'Joining' tables together based on data values. There is no need to use conventional programming techniques to obtain this information - just one relational request. The power of SQL is that it provides operators which can process sets of rows - rather than needing to use single row at a time processing. A single relational request can thus be used selectively to retrieve, update or delete multiple rows of a table. This is called 'set processing'. In conventional programming language approaches, such operations would require multiple requests. As an example - if we asked for details of all employees working in department 'D11' from the **EMPLOYEE** table, we would be presented with the result in the form of a table with two rows in it, containing details of the relevant employees.

So a relational database is one which presents tables of data to the user, but which also provides the full power of relational processing to the user via SQL.

Table 2: SQL - A Relational Language

Many relational databases make use of the Structured Query Language, SQL. SQL is a non-procedural language. Software developers and users specify what they want, not how to do it. In our example, if we wanted to know the **NAME** and **SALARY** of EMPLOYEE 60, we would enter:

```
SELECT NAME, SALARY
FROM EMPLOYEE
WHERE EMPNUM = 60
```

We would receive:

NAME	SALARY
Henderson	8000

SQL supports all the functions one would expect in a relational language, such as 'Joining' tables, and 'Set processing'. As another example, to give everyone in department 'D11' a 10% pay rise, we would enter:

```
UPDATE EMPLOYEE
SET SALARY = SALARY * 1.1
WHERE DEPT = "D11"
```

SQL has built-in functions and arithmetic operators, which allow users to group or sort data, and find the average, minimum, maximum and total of a specific column.

SQL is also a multipurpose data language. The same language is used to define, retrieve and manipulate data. As an example, to create the **DEPARTMENT** table, all that is required is to type in this SQL statement:

```
CREATE TABLE DEPARTMENT
(DEPTNO CHAR (3)
DEPTNO VARCHAR (20));
```

Once the table is created, data can immediately be entered into it. Table design and creation is very simple. But, even more important, is ease of table modification. Columns can be added to a table by keying a single statement. No reorganization of the data is needed.

A key feature of SQL is that the same syntax is used by end users constructing queries as programmers writing applications.

* A user can address all information as a value within a table, not by its physical position within that table.

* The database provides a user-transparent data navigation mechanism – ie the system, not the programmer, is responsible for determining any necessary storage/retrieval access paths.

* The database performs operations on data at the set level (in the mathematical sense). Any one operation is performed on a set of records. Similarly, any one operation results in the creation of a new relation.

Again, according to Codd, any truly relational database implements all the above features of the relational model – and a couple of slightly more complex ones. He says that the use of a relational database provides for a number of benefits including data independence at both the physical and logical levels, automatic navigation in gaining access to data and support of logical views of data. Perhaps more relevant are the ability to use high-level languages, such as SQL, to manipulate data, increased programmer productivity and reduced cost and time in applications development. Other benefits include ease of access to data in order to perform ad hoc data queries and improved data integrity (see 'Implementing Referential Data Integrity' in the October 1987 issue of .EXE).

Relational Database Rationale

There are two basic reasons why relational databases are easy both to understand and use. The first is that they are based around the concept of 'tables' of data, whose contents reflect, and are defined by, 'the key of that table, and nothing but the key'. In essence, these tables represent basic building blocks of data. They can be merged, analysed and cross-referenced as a means of constructing complex views (ie reports, or on-line displays) of one or more the these tables. Furthermore, because of the formal, mathematically-based nature of relational theory, this merging and cross referencing of data can be performed using a limited, rigorous, and easy to use set of processing commands.

The second reason relates to the manner in which a program 'navigates' around a relational database. It doesn't. It simply requests, via the use of SQL, sets of information by means of key fields. The database does the rest, returning the required information when found. Contrast that with other databases, where a program must explicitly navigate between different database elements. The programmer must, for example, explicitly code 'first find the master record, now the invoice, now the product from the invoice, now the next invoice', and so on.

As it happens, the coding is not totally dissimilar much the same in each case. But, with non-relational products, the structure of the resultant program is tied in absolutely to the structure of the database. Change the database structure, and you're almost certain to have to change the program as well. With a relational database, the program doesn't even know the database structure, so will not need to be changed when that structure changes.

The Performance Question

The very ease of use of relational databases has already made for a number of myths, primarily that relational systems are only good for ad hoc querying and do not meet performance requirements for production systems under transaction processing. Another myth says that relational systems require a future hardware breakthrough such as associative memory to achieve their performance promise.

Certainly, anyone considering a relational database needs to take the performance question seriously. Older style databases perform well because they have physical links built into their data structure. Relational database eschew that approach as an act of faith, and the price they pay is relatively poorer performance. But the performance problem is being cracked. A large number of techniques are being used by suppliers to increase the speed of

their products. For example, many products hold closely related data from different relational tables in the same physical disk space. Another common method of improving relational performance is that of deferring database updates until the end of a logical event.

Other more complex methods have also been implemented which optimize the use both of relational database disk space and which improve the flexibility or relational indices. These latter methods include some degree of 'pre-joining' separate tables (ie overlaying built in logical links between tables), allowing, for example, one access to the customer table index, to provide access both to that customer's order and invoice details.

When people criticise relational systems for lack of performance, they usually fail to take into account the abundance of functionality that relational systems offer. As just one example, a relational system may perform multiple query functions in a single request, while a traditional database may require several different processing requests – and the accompanying system time – to answer the same request. The reality now is that there is no intrinsic reason why a relational system should not perform as well as, if not better than, a traditional database under transaction processing conditions. Indeed, most sales of 'performance' relational products, such as Adabas, are made on the basis that the computer system they will support will be critical, operational ones.

"SQL supports all the functions you would expect in a relational language and is a multi-purpose data language"

Recent Developments

Within many environments, relational databases have already gained a firm and longstanding foothold, inevitably, relational products are being extended to cover other processing requirements. One trend provides the ability to store different types of information within the relational framework. For example some products support the definition and storage of 'irregular' data types, with such a product, a developer might define 'picture' as a data type and query it in the specialised context of the application by entering 'select any picture that contains a circle'.

Perhaps the hottest topic in relational databases at the moment is distributed databases. These allow users to access data at a number of different physical locations, without needing to know anything about where the data is actually located. The glue holding distributed databases together is SQL, and we shall be returning to this topic later in the series. However, the next article will look at the history of SQL, and at the facilities it offers the software developer.

Russell Jones is a freelance journalist and a regular contributor to .EXE. He can be contacted on 0954 80358. His SQL series will run for a further 5 issues. Next month's article gives a functional overview of SQL.

Figure 1: Two Tables from our Sample Company Database

EMPLOYEE TABLE

EMPNUM	NAME	DEPT	SALARY
60	Henderson	D11	8000
70	Smith	D12	7000
80	Jones	D11	6000

Related by DEPT/DEPTNO

DEPARTMENT TABLE

DEPTNO	DEPTNAME
D11	Accounts
D12	Sales

RT²

Gives you real-time on a pc

RT² provides multitasking, timing, memory allocation, semaphore, and i/o support for real-time applications. It is fast, modular, compact and configurable, and fully documented. Using RT² you can get the following benefits:

- reduced costs and improved performance
- high performance interrupt-driven code
- support for both Microsoft and Intel high-level languages
- flexible licensing to suit small or large users

Enquire now:

Alan Cohen (Computer Consultants) Ltd
225a Finchley Road
London NW3 6LP
Tel: 01-435 5493

Alan Cohen

CIRCLE NO 797

ARTEK/ADA ARTEK/ADA ARTEK/ADA

Artek/Ada is a very full (and soon to be validated) Ada Systems. It runs on an IBM PC in 384k of RAM with MS-DOS or PC-DOS 2.0. The Artek System has an APSE (Ada Program Support Environment) which enables you to easily edit, compile, link and run your programs and manage your Ada library. The compiler produces A-code which is run using a powerful debugging facility or converted to the native code of your micro. Use the language of the 1990's, now on your micro.

ADA SCIENTIFIC LIBRARY

A PC Library covering the main areas of scientific and mathematical computing includes basic maths functions, complex numbers, random numbers, array operations, linear equation solving and sorting.

CATALYST CATALYST CATALYST

CATalyst is an on-line Ada Encyclopedia. Available on your screen, as and when required - explanations of Ada terms and features, dynamic simulations of Ada programs etc, with instant cross referencing. A hard disc is recommended.

ALGOL 68 ALGOL 68 ALGOL 68

Algol 68 is perhaps the most elegant language invented. Its orthogonal design makes it so easy to use, yet it is one of the most expressive languages available. This powerful subset of ALGOL 68 from Algol Applications is available as an interactive or an interactive and batch system combined. It fits easily in a 256k IBM PC or compatible.

ALGOL 68 MINI-LIBRARY ALGOL 68 MINI-LIBRARY

Twenty routines in Algol Applications Algol 68, covering the main areas of numerical computing - integration, linear algebra, random numbers, linear programming and sorting.

CHIWRITER CHIWRITER CHIWRITER

At last, a full mathematical word processor. Greek/mathematical symbols, super/subscripts, underlining, many fonts, etc. All displayed clearly on your screen. Drives most matrix and laser printers.

Please call or write for further information

Artek/Ada (new low price)	£350
Artek/Ada Demo (deductible from price of full system)	£20
Ada Scientific Library	£120
CATalyst	£640
Algol Applications Algol 68 Interactive System	£175
Interactive and Batch	£350
Algol 68 Mini Library	£50
Chiwriter (INTRODUCTORY OFFER)	£69

Prices are for single machine usage. Other prices on request. Educational Discount 10%. Please add 15% VAT and £3 p&p to orders (no p&p on Artek/Ada Demo).

N.A. SOFTWARE LIMITED

Numerical and Computational Consultants
Merseyside Innovation Centre, 131 Mount Pleasant, Liverpool L3 5TF Tel: 051 708 0123

DEALER ENQUIRIES WELCOME

Ada is a registered trademark of the U S Government AJPO. Artek is a trademark of Artek Corporation

CIRCLE NO 798

System Science

C COMPILERS

AZTEC	
AZTEC C86/PROF (MS-DOS) large mem, assm, link	£149.00
AZTEC C86/DEV (MOS-DOS) adds source debug, editor	£225.00
AZTEC C68/COMM (MOS-DOS) adds ROM support, lib source	£325.00
AZTEC C68/COMM-MAC, AMIGA	£325.00
AZTEC C for PC/M-80, Apple Dos, Prodos	£ call

LATTICE	
LATTICE C Compiler ver 3- 8087 all mem models	£235.00
LATTICE Screen Editor	£95.00
LATTICE C SPRITE Debugger	£129.00
LATTICE dbC-II or III Dbase library	£175.00

MICROSOFT C 5.0 with Code View, Quick C large mem, source debugger, 8087	£325.00
TURBO C from Borland **NEW** Editor, Linker Large mem options	£69.00
LIVING C PLUS (NEW) Interpreter and Comp	£145.00
RUN/C-PROFESSIONAL Interp.	£125.00
RUN/C — Interpreter excellent tutorial	£60.00
C CROSS Compilers 6809, 68070, 8051, 6301, 80386	£ call

C LIBRARIES

GRAPHICS	
Metawindows — many languages	£115.00
Essential Graphics Library	£175.00
GraphiC — source, colour, EGA	£245.00
Halo — specify compiler	£195.00
SCREEN and DATA ENTRY	
PANEL entry screens — most languages	£215.00
Vitamin C — Screen, print layout code	£145.00
Windows for Data, Windows for C	£215.00
ISAM & DATAFILE	
C TREE — Faircom (source) B tree lib.	£245.00
R TREE — Reports generator for C TREE	£195.00
Btrieve — database library, many lang.	£195.00
Btrieve/N — Novell, PC NET networks	£425.00

COMMUNICATIONS	
GREENLEAF COMMUNICATIONS Ints, Ring Buff, Status & Control	£115.00
BLAISE ASYNCH MANAGER Port control, XMODEM protocol	£125.00
GENERAL	
GREENLEAF GENERAL FUNCTIONS Dos, Disk, Video, Strings, Date, Keybd	£115.00
PforCe — source lib, comms, dbase, screen etc The C programmers toolkit	£225.00
BLAISE C TOOLS PLUS source DOS, Dos screen, windows, keyboard etc	£125.00
C Scientific Subroutines — source Matrix, polynomial, diff eqn etc	£125.00

PHOENIX	
PLINK-86 Plus — overlay caching	£295.00
PMATE-86 programmers editor	£120.00
PRE-C Lint utility	£195.00
PFIX-86 Plus debugger	£225.00
PDISK - backup, disk cache & utils	£95.00

BRIEF multi-file editor	£145.00
dBRIEF addition for DBASE programmers	£75.00
EPSILON — Emacs style editor	£145.00
PC/VI screen editor	£125.00
MKS Toolkit lots of Unix goodies, inc VI	£95.00
QUILT Software Revision Management	£95.00
PC-LINT by Gimpel	£99.00
CLIPPER — DBase III Compiler	£545.00

ASSEMBLERS and CROSS-ASM

Microsoft 8086 macro Asm (SYMDEBUG & LINK)	£115.00
Cross Assemblers 68xx, 68000, Z80, 8080, 6502, 8048, 8051 etc	£ call
Simulators - Z80, 8048, 8051 etc	£ call
Quelo 6800 and 68020 cross assemblers	

LMI Forth-83

PC-FORTH, asm, Editor, DOS access	£115.00
PC/FORTH Programmers package with 8087 graphics	£175.00
UR-FORTH high perf., 8087, graphics	£245.00
METACOMPILRS 8080, Z80, 8096, 6303 etc etc	

BORLAND	
TURBO C (NEW)	£69.00
TURBO PASCAL	£69.00
TURBO PASCAL JUMBO PACK	£225.00
Turbo Tutor	£29.00
TURBO PROLOG new	£69.00
TURBO LIGHTNING	£69.00
Superkey or Sidekick	£69.00

TURBO DATABASE TOOLBOX	£49.00
TURBO GRAPHICS TOOLBOX — PC-DOS	£49.00
TURBO EDITOR TOOLBOX — PC-DOS	£49.00
BLAISE POWER TOOLS PLUS	£75.00
BLAISE ASYNCH PLUS	£75.00
TURBO POWER UTILITIES	£75.00
TURBO POWER EXTENDER	£75.00
T-DEBUG PLUS	£49.00

FORTRAN/PASCAL/BASIC

Fortran Graphics, Scientific Libraries	£ Call
Microsoft FORTRAN-77 with Codeview	£275.00
Pro-FORTRAN-77	£395.00
RM-FORTRAN-77 with RM-FORTE toolset	£595.00
PRO-PASCAL MS-DOS	£295.00
Microsoft Pascal	£185.00
QUICK BASIC — Microsoft (PC-DOS only)	£75.00
PASCAL 2 full features, Turbo compatible	£325.00

LISP and PROLOG

TURBO PROLOG (PC Only)	£69.00
micro-PROLOG PROF (full mem., wind.)	£245.00
micro-PROLOG entry level version	£95.00
MuLISP/MuSTAR (Common LISP)	£195.00
Golden Common LISP (PC DOS only)	£495.00
MuMATH/MuSIMP Symbolic maths	£275.00
TransLISP Common	£75.00
PC Scheme from Texas	£95.00

TEXT WINDOWS, COMMUNICATION, DISK

VENTURA Desktop Publishing System	£695.00
FINAL WORD 2 author's WP	£275.00
MicroTEX — scientific typesetting	from £350.00
MICROSTAT — comprehensive statistics	£295.00
STATGRAPHICS statistics and graphics	£545.00
WINDOWS by Microsoft	£85.00
WINDOWS Software Dev. kit	£345.00
ZAP Communications - VT100, 4010 emulation	£85.00
PC/Intercomm — VT100, 102 Kermit etc	£95.00
UNIFORM — PC r w formal CP/M disks	£65.00
CONVERT PC add formats	£89.00
UNIDOS run CP/M software on PC	£65.00
NORTON UTILITIES advanced	£99.00
Dan Bricklin's DEMO program	£65.00

MODULA-2	
Pecan Modula-2 Powersystem	£95.00
Pecan Modula-2 Advanced Pak	£195.00
Logitech Modula-2/86 Apprentice	£69.00
Logitech Modula-2/86 Wizards package	£159.00

PECAN ucsd PRODUCTS	
POWER SYSTEM p system with a compiler Pascal, Fortran-77, Basic or Modula-2	£95.00
PDQ Pascal starter pak	£69.00
JACK-2 Integrated WP, database, spreadsheet	£95.00

All prices are exclusive of VAT. Please add £3.00 p&p, plus VAT to your order

6-7 West Smithfield, London EC1A 9JX
Tel: 01-248 0962
BTGOLD 76: CJJ028

ACCESS
BARCLAYCARD VISA

CIRCLE NO 799

Adding Better EGA Support to MS-DOS

Although the EGA card lets you have 43 lines of text on screen, DOS insists on scrolling after 25. Robert Schifreen presents the solution.

The Enhanced Graphics Adaptor and matching monitor, introduced early in 1985, scored a number of points over the CGA. Text quality was vastly improved and maximum graphics resolution increased to 640x350. Two character fonts were provided in ROM, though it was also possible to design entirely new fonts simply by loading the data into memory and using an INT 10H call load them into EGA card's RAM.

The EGA card contains its own BIOS ROM, which inserts its address into the machine's INT 10H vector at boot time. This ROM takes over the normal BIOS video services that plot points, set screen modes, draw characters and so on. Incidentally, the original INT 10H vector is copied into INT 42H by the EGA card's startup code, so that the card can still access the machine's own INT 10H functions by issuing an INT 42H command.

Although the video BIOS services are enhanced by adding an EGA card, the DOS functions are not. This is often not a problem, because DOS services tend to work by calling the BIOS, so the EGA's new BIOS will affect DOS calls, too. The main time that DOS becomes a problem is when you put the EGA card into 43 line mode. A number of programs do this, for example LIST and PC-Write. As long as the application knows that it is currently operating in 43-line mode, all will be well.

Using a public domain utility like EGA43, you can select 43-line mode from the DOS prompt. If you want to go one better, there's something called 8X6.COM that will actually give you 60 lines. There's a problem with these programs, though. MS-DOS (and PC-DOS) doesn't know that the number of screen lines has changed. Therefore, DOS will insist on scrolling after 25 lines, effectively leaving the lower half of the screen blank. Also, the CLS command will only clear 25 lines. After putting up with the problem for some time, I decided that it needed investigation. I studied my PC's BIOS, and the one on the EGA card, looking for clues. Nothing. Further digging proved that the culprit is in fact the CON device driver. When MS-DOS wants to output a character on the screen, it opens a channel to the console device driver, sends the character, and then closes the channel. It's the device driver that actually puts the character on the screen.

The code for the driver is normally buried in IBMDOS.SYS – a hidden file loaded during the boot process – but is overwritten if the ANSI.SYS driver is used. Examining it reveals that BIOS calls – one level below DOS and the driver – are ultimately used to put characters on the screen. It also reveals that a screen length of 25 is hard-coded into the driver. Patching IBMDOS.SYS is not a good idea. For a start, the boot process locates the file not by name, but by its position on the disk. Therefore, you can't just unhide the file, copy it and patch it. Also, you'd have to redo the patching if you changed to a different version of DOS. What's needed is a way of patching ANSI.SYS, to give a portable solution to the 25 line problem.

Like the rest of the PC's controlling software, the EGA BIOS uses segment 0040H as a data area. At offset 0084H it stores a byte that indicates how many lines the current screen mode is capable of displaying. If the byte is zero, the screen is set to 25 line mode. Otherwise, the byte is one less than the number of rows that can be displayed.

All mode-changing software that I have come across keeps this byte updated, so what I finally did was to patch a version of ANSI.SYS that, instead of assuming that the screen displayed 25 lines, would read the byte at 0040:0084H to get the screen depth. Microsoft probably wouldn't be too happy about my printing the entire source code of the new version, or even offering it on disk as a binary file. Therefore, the following information explains how to do the patch yourself. You will end up with a version of ANSI.SYS that works properly not just with a 25 line display, but also adapts itself for use with 43 and 60 lines. I'll leave it up to you to get hold of the utilities that actually change the modes.

All you'll need is a copy of ANSI.SYS and DEBUG. If you know nothing of IBM PC assembly language or DEBUG, you may have problems. However expert you regard yourself, make sure you have a bootable floppy disk handy, with which you can recover the machine in case you make a mistake and finish up with loss of display functions.

Extending the program

My version of ANSI.SYS was around 1900 bytes long. You'll need to make the program around 100 bytes longer, so that you can stuff some subroutines at the end of the code and then CALL them from within

the driver. You can't insert the patches in situ, because loading a register with the contents of location 0040:0084H takes more bytes of machine code than simply loading it with an immediate value of 25. To extend the file, type:

```
DEBUG ANSI.SYS
```

and, at the hyphen prompt, display the registers with the **R** command. Register CX holds the length of the program, to which you can add some bytes just by adding something to the value in CX and writing the file back to disk. I made my version exactly 2000 bytes long by setting CX to 7D0H. Do this with the **RCX** command, and typing the new value for CX when the colon prompt appears. Write the new, longer program back to disk with the **W** command, and check that its length has changed.

Changing the Line Count

The first command that must be patched is around 278H. Look for code something like Figure 1.

I've given the hex values so you can use DEBUG to search for the patterns. The CMP command is checking which line the cursor is on. If it's on line 25 (19H) then the screen will be scrolled. Using the **A** command in DEBUG, change the **CMP** command to a **CALL**, to call a subroutine in the space you have created at the end of the program. The **CALL** instruction will be 3 bytes long, so make sure that you put two NOPs after it, otherwise you'll have a spurious 2-byte instruction that will crash the system. The subroutine, instead of comparing the byte at location 102H with 19H, must compare it with the contents of location 40:84H. Further, the flags register must be preserved so that the JB will work, and we're going to have to explicitly set DS to segment 40H. The routine is Figure 2.

These are the commands exactly as they are typed into DEBUG's assembler. Note that all numbers default to hex, and that the **DS:** segment override is a separate command. Make a note of where your routine ends, so you'll know where to start the next one. Leave at least 6 NOP's between each routine, though, just in case.

The Cursor Return Line

The next command to be patched starts at around 27FH. The code looks like Figure 3

The **MOV** instruction tells the driver which line the cursor should return to after the screen has been scrolled. This is normally hard-coded to 18H, which is line

Figure 1: The Old Line Count Routine

```
06              PUSH ES
0201            ADD  AL,[BX+DI]
803E020119      CMP  BYTE PTR [0102],19
7208            JB   0287
```

Figure 2: Changing DS

```
PUSH  AX          ; preserve AX
PUSH  DS          ; and DS
MOV   AX,0040     ; This is the segment we need
MOV   DS,AX       ; so put the value into DS
DS:               ; the next instruct reference DS
MOV   AL,[0084]   ; get the no. of screen lines -1
INC   AL          ; now it really is no. lines
POP   DS          ; don't need DS any more
CMP   [0102],AL   ; the new version of CMP
POP   AX          ; we don't need AX now, either
RET               ; return, to let JB take over
```

Figure 3: The Old Cursor Return Instruction

```
C606020118      MOV   BYTE PTR [102],18
E81100          CALL  2098
```

Figure 4: The Replacement CALL Routine

```
PUSH  AX          ; preserve AX
PUSH  DS          ; and DS
MOV   AX,0040     ; This is the segment we need
MOV   DS,AX       ; so put the value into DS
DS:               ; next inst must reference DS
MOV   AL,[0084]   ; get the no. of screen lines -1
POP   DS          ; don't need our DS any more
MOV   [0102],AL   ; put value where ANSI wants it
POP   AX          ; we don't need AX now, either
RET               ; all done
```

Figure 5: The New CLS Subroutine

```
PUSH  AX          ; preserve registers
PUSH  DS
MOV   AX,0040
MOV   DS,AX       ; set data segment to 0040h
DS:
MOV   AL,[0084]   ; get screen depth value - 1
INC   AL          ; correct it
POP   DS
MOV   DH,AL       ; the revised MOV command
POP   AX
MOV   DL,[0100]   ; the MOV we had to move!
RET
```

Figure 6: The Old Scrolling Instructions

```
BEA000          MOV  SI,00A0
B98007          MOV  CX,0780
FC              CLD
2E              CS:
813E170100B8    CMP  WORD PTR [0117],B800
```

Figure 7: The New Screen Routine

```
PUSH  AX
PUSH  DS
MOV   AX,0040
MOV   DS,AX
DS:
MOV   AL,[0084]   ; get screen depth value - 1
POP   DS
MOV   AH,50       ; 50h = 80 decimal
MUL   AH          ; multiply screen depth by 80
MOV   CX,AX       ; put result where ANSI expects
POP   AX          ; tidy up
RET               ; and we're done.
```

24. We want to change this to the value of 0040:0084H. The **CALL** instruction will again be 3 bytes long, while the original **MOV** was 5, so don't forget the NOPs. The routine to be called is:(see Figure 4)

Improving the CLS command

For some unknown reason, the CLS command is also handled by the console driver. Luckily for us, it means that ANSI.SYS is also the place to patch, to make sure that a full 43 or 60 lines are actually cleared. Around location 584H you'll find:

```
33C9        XOR   CX,CX
890E0101    MOV   [0101],CX
B619        MOV   DH,19
8A160001    MOV   DL,[0100]
8A3E1501    MOV   BH,[0115]
```

We need to replace the **MOV DH,19** with a routine to move the correct value into DH. Unfortunately, the **MOV** above is only 2 bytes long, and we need to replace it with a **CALL** that takes 3 bytes. The solution is to replace not just the **MOV DH,19** but also the **MOV DL,"0100"**. This gives us 6 bytes, of which 3 will be the **CALL** and 3 will be NOP's. Both commands will then be handled in the subroutine instead. The code for the subroutine is:(see Figure 5)

There's only one more command to change, though it's the most tricky. Somewhere near location 02BDH will be: (see Figure 6)

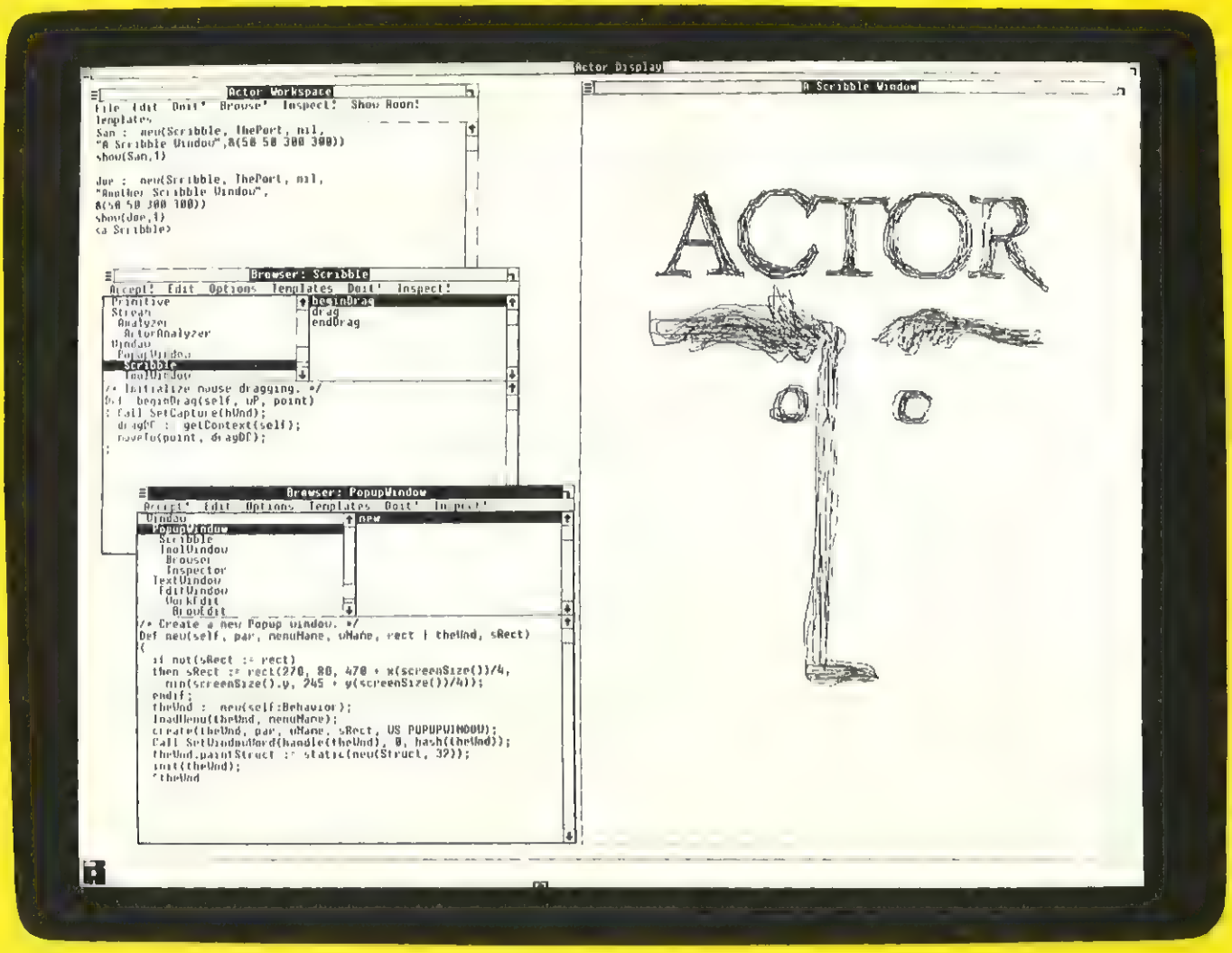
"The culprit is the CON device driver where a screen length of 25 lines is hard coded: you need to patch ANSI.SYS"

The actual screen scrolling is done by a block-move of the contents of the video RAM. The value of 0780H, which gets moved into the CX register, is the number of bytes that have to be shifted up the screen. The value of 0780H is 1920 decimal, which is 24x80 (24 lines of 80 characters). The new routine must work out the new value by multiplying the screen depth byte by the number of characters per line. Although the byte at location 0040:044AH actually stores this value, few programs ever change it. The routine, therefore, will also assume that the screen is displaying 80 characters per line.(see Figure 7)

That's it. Save your new version (I call mine ANSI-EGA.SYS) and try it by putting DEVICE = ANSI-EGA.SYS in your CONFIG.SYS file. Check that the screen scrolls properly, that **CLS** works, and that the scroll happens at the right time. Despite the extra subroutines, you'll not notice any loss of speed. Finally, if you don't have an EGA monitor, or you don't have a utility to set the screen depth to 43 lines, you can still have fun by changing the byte at 0040:0084H and making screen sizes of less than 25 lines.

Robert Schifreen is .EXE's Deputy Editor, and will become the magazine's editor as from January's issue.

HOW TO WRITE A WINDOWS APPLICATION IN TEN MINUTES.



Actor™ is a new language that combines Microsoft® Windows with object-oriented programming. This means you can produce mouse and window applications very quickly.

For example, we created a simple "paint" program, and used it to draw the Actor logo you see on the screen. The whole program only took ten lines and ten minutes. Part of it is in the middle window on the left.

Above, you see the commands that initialized the paint window and made it appear on the screen. Below, some code that's built into Actor, specifying window behavior. Through a process known as "inheritance," it's called into play automatically.

Try programming in this new way, and you'll never go back.

NEOW **The Windows Specialists™**

NEOW. 470 London Road, Slough, Berkshire. SL3 8QY. (0753) 45115

Name _____ Position _____

Company _____ Address _____

_____ Postcode _____

Phone _____ Business _____ PC Type _____

Please send me more information about Actor ☐ and the Neow Windows Catalogue ☐

CIRCLE NO 801

.EXE Binder

At last, we are pleased to be able to supply readers of .EXE with an exclusive binder to keep a full year's supply in pristine condition, for easy display, quick reference and pure nostalgia. Simply fill in the form below.

Please rush me____.EXE binders at £8.50 each inc. Post and Packing
Send_____blue and_____ grey binder. Total order value £_____
Please debit my VISA Card No:_____Exp. Date:_____
I enclose a cheque.
Name:_____Tel:_____
Address:_____
_____Postcode:_____
Send this form now to .EXE Magazine, 10 Barley Mow Passage
Chiswick, London W4 4PH

Coward on the 68K

'The oh thirty marks the complete commercialism of 32-bit computing', the words of James Norling, Executive Vice President and General Manager of Motorola Semiconductors. They have done it, they have finally done it, the 'oh thirty' (a nickname acquired by the MC68030 in a burst of passion at Motorola) has finally been launched. It seems the major players in the VME industry have gone oh thirty mad, various people are claiming that they have an oh thirty board up and running, but they are rarely seen to do so. The popular place for an oh thirty board to be mounted does as yet not appear to be on a VME chassis but instead a glass cabinet or shelf.

Following my request to discover where Plessey got their stenciling kit from, I was contacted by Top Brass at Plessey (who will remain anonymous), to be told quite matter of factly that they got their chip from Motorola on account of it being dead (well not in so many words) and that they had not lashed up a homebrew lookalike. .

There is also a lot of backstabbing going on, with people accusing other people of cobbling an oh thirty into an 020 board. Apparently this is bad practice as it does not allow the cache burst mode of the oh thirty to be implemented, thus drastically impairing the oh thirty's performance almost down to the level of an 020. We all know that the cache handling of the oh thirty is one of the things that make it great. You see, according to my sources the important thing is not to have the first working oh thirty board but the best working board (after all it is not whether you win, but how you play the game).

In October, there was talk of a standard real-time operating system with UNIX extensions. This month Motorola is conspiring to bring about a standard UNIX, compatible at the binary code level in conjunction with the UniSoft Corp. This development is due to the high integration of the MC68030 requiring less custom hardware to be used thus making memory handling standard, regardless of the who supplies the system. The 68030/UNIX Binary Portability Standard is currently available for review to all interested parties, and Motorola and Unisoft will consider additional amendments during 1987. A formal standard will be announced in early 1988.

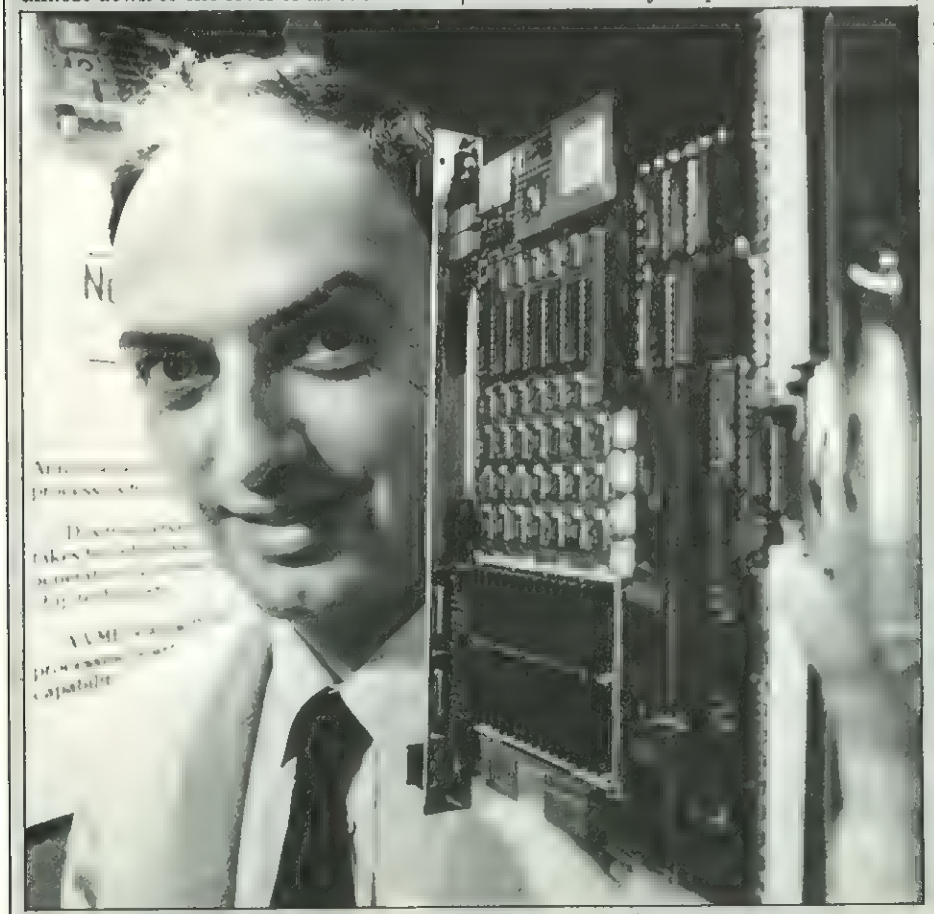
Now some goodies for all you software developers and system builders raring to go with the MC68030. Motorola have announced an optimised C compiler as well as an emulator module. The C compiler was written to run under System V.3.1 but is said to be portable to other UNIX implementations. The emulator module (HDS-300) is a host independent interface that connects any computer to the oh thirty emulation module; this allows concurrent use of the emulator. To further enhance the performance of the MC68030 there will be new floating point co-processor named the MC68882, but it appears that Motorola were not significantly impressed to nickname the product so we will just call fast.

As regards to further spiced up 68Ks, a 25MHz version of the MC68030 is due for sampling in December so hold on to your hats. It is rumoured to provide up to three times the performance of the previous MC68020. There are plans to build a 30MHz version and beyond (the fastest MC68020 was rated at 38MHz but was never mass produced). But for those of you out there who have already spent a great deal of time and effort developing your own memory management hardware (I know who you are, and so do you) there is good news for you too. Motorola are planning the 040 which will upwardly compatible and offer significant performance increases over previous 68000s. This is all anyone is willing to tell me, its almost as if Motorola are not quite sure what else to add to their chips.

But enough about Motorola's new baby, what if you're quite happy with your current chip? Soft PC synthetic hardware, is a product providing a MS-DOS environment in software for MC68020. The package is available for the Mac II, Intergraph's Interpro 32C, Tektronix 4300 series, Sun-3 systems and will soon be available for other machines. PC XT performance is achieved on 68020 system with PC AT performance on a 68030 system. The main components are a CPU module, an I/O devices module, a memory module and a BIOS module. The product is compatible with products like MS-Windows, Sidekick, Flight Simulator, Dbase III+ and Turbo Pascal. Phoenix, famous in the PC world for their BIOS implementations, have also announced a software based MS-DOS emulator for the 68000 family. They also claim that they pioneered the concept... Intel 8087 support is implemented via the numeric co-processor, and an MC68020 running at 25MHz is rated at 6.0 Norton SI points.

If any of you have come across any PC emulators, call in and tell me what you think. If you have a 'working' oh thirty I would love to see it. By the way I did try to get you a picture of an oh thirty but to no avail, but we have been promised one we can use in the mag. In the mean time you can all envy me because I have a picture of an oh thirty on my wall.

- Hobbit Coward



Plessey's Barry Mallett: "No stencil kits at Plessey"

Promotional Reprints of Articles Featuring Your Product

Considering the fact that other people's independent opinions endorse the value of your product, .EXE would like to offer you the opportunity of ordering reprints of .EXE articles which feature your product or company, for you to include in your promotional mailings, sales packs and press releases.
The articles may be adapted especially for you, to include your company logo, more tables and diagrams, further information etc.

For more information about this ideal promotional opportunity, call Helena Adams on 01-994 6477 ext. 359.

Minimum quantity: 200 reprints.

ADVERTISERS INDEX

INTEGRATED S FTWARE FOR BUS-BASED SYSTEMS... AND STANDALONE APPLICATIONS

APPCOM – A Streamlined and cost-effective approach to Target System development:

1. **NO NEED FOR DEDICATED DEVELOPMENT SYSTEM** – use any low cost PC!
2. **CHOICE OF MANY STANDARD PC COMPILERS:** e.g. Turbo Pascal, BASIC, 'C'.
3. **FEWER DEVELOPMENT STEPS** – faster development cycle!
4. **DEBUGS THE TARGET, NOT THE HOST!**
5. **EXTENSIVE CHOICE OF 3RD PARTY TARGET HARDWARE,** ready-made or custom, e.g. STE.
6. **RELIABLE, PREDICTABLE PERFORMANCE AT LOW, CONTROLLABLE COST.**

APPCOM enables programs written and tested on a PC to be downloaded to the Target's RAM for development/debug in the Target using the PC's screen, keyboard and disks. The final debugged program is copied from RAMdisk to PROM leaving the application to run in the target independently of the PC.

APPCOM is a complete software engineering system that supports MS-DOS/PC-DOS system calls. It is not simply MS-DOS in ROM.

Streamline your next project with LOWER CAPITAL COSTS, NO LANGUAGE BARRIERS, LOWER DEVELOPMENT COSTS and PREDICTABLE RESULTS! For a Demo disk contact:

Hexatron Ltd Unit 5 Business Centre, Avenue One, Letchworth, SG6 2BB.
Tel: 0462 675530.

hexatr n **Embedded Computing**

Trademarks: IBM-PC-DOS are trademarks of IBM Corp, MS-DOS of Microsoft Corp, Turbo Pascal of Borland International. Appcom Software emulates MS-DOS/PC-DOS and Hexatron own the full copyright.

CIRCLE NO. 803

THE .EXE OFFER

25% Discount on a C Code Generator

An exclusive discount for .EXE Readers: PRO-C and PRO-C Workbench available on MS/PC- DOS, XENIX and QNX for users of Borland C, Microsoft C, DeSmet C, Lattice C, Zorland C, or Computer Innovations C86 -

In this month's .EXE offer, we are pleased to be able to offer 25% discount on these new products from the States. PRO-C is a C source code generator which produces tailor made, stand alone programs and writes fully optimised, commented and documented programs. The package consists of the following fully integrated generator modules:-

* Record Definition
* Screen Generator
* Update Generator
* Menu Generator
* Report Generator
* PRO-C Painters

Requirements:
MS/PC DOS V2.0 (or later versions) or QNX or XENIX
256k RAM
Hard Disk
5.25" Floppy Disk Drive
One of the following C Compilers:- Microsoft C, DeSmet C Lattice C. Computer Innovations C86, Zorland C, Turbo C

The complete package is offered to .EXE readers for just £685.69 (normally £914.25), or individually:- £547.00 for PRO-C (£714.00), £317.40 for Workbench (£396.75)

PRO-C uses the powerful and efficient ISAM File Handling Technique; BTRIEVE and C-ISAM are supported and PRO-C can interface with any other ISAM Handler. Generated programs do not require a run time license neither does PRO-C have to resident on the host machine after creation of programs. Unlimited on-line help can be incorporated in any program generated.

To help you get the very most from PRO-C we have made available PRO-C Workbench - an individual productivity tool for any C programmer. Workbench is a set of over 60 Library routines (which comes complete with all C Source Code) used both in the PRO-C products and in the programs you generate with PRO-C.

Please send me:-
_____ copies of the combined package with a 25% discount -
_____ copies of PRO-C with a 20% discount
_____ copies of Workbench with a 20% discount

I enclose my cheque made payable to .EXE Magazine for £_____
Please debit my VISA Card no_____Exp. Date_____

Name:_____ Job Title:_____
Address:_____
_____ Post Code:_____

Please add £1.50 postage and packing. All prices quoted are inclusive of VAT.

CIRCLE NO 804

PRODUCTS:

MS-C For Apollo Computers

The OASYS Microsoft Cross C Development System has been announced by Apollo in conjunction with OASYS, Inc, for Apollo's range of UNIX based workstations. This offering lets software engineers use both Apollo workstations and Microsoft C to develop MS-DOS applications to run on PCs. The development system is a complete set of tools that include the Microsoft C Compiler, Macro Assembler (MASM) and Linker, all necessary components for a complete PC cross development solution. Apollo's Domain Environment is well suited to PC development. The Domain PCI PC Interconnect permits easy transfer of data and files between PCs and Apollo systems. Domain/PCC, Apollo's 80286 IBM PC-AT compatible coprocessor, and Domain/PCE PC-AT software emulator, both provide a PC/MS-DOS window on Apollo workstations. Apollo's PC compatibility products allow developers to use the same workstation for the entire development cycle – from designing, coding, and compiling, through to the testing of MS-DOS applications.

Circle No 760

Embedded SQL for Ada Applications

INFORMIX-ESQL/Ada, a product that allows Ada programmers to use SQL from within their Ada source code has been released by INFORMIX Software. The product was created to meet the growing demand for database management and application development software on ANSI-standard SQL for Ada. With ESQL/Ada, developers embed SQL syntax into Ada code to create an application that interfaces with Informix's SQL database engine. Embedding SQL statements are designed to build and manipulate database files. This enables developers to specify a desired result in many fewer lines than in Ada. Users can achieve a high degree of application customization and flexibility in development through its dynamic query capability. In most applications, the queries that users are likely to ask are predetermined and embedded into the program. With dynamic queries, users can formulate ad hoc queries , to most effectively access and manipulate the data. ESQL/Ada is compatible with all other Informix software products, including those that support the TCP/IP networking standard. Initial shipments of INFORMIX-ESQL/Ada in November, and will support Ada Compilers from Alsys Corp. and Verdix Corp. on Sun 3 workstations running under the UNIX operating system. INFORMIX have also announced the a SQL relational database management system running under A/UX on the Apple Macintosh II which will be available with the first shipments of the Mac II.

Circle No 761

MS-QuickBASIC V.4.0

Microsoft QuickBASIC Version 4.0, a combined compiler-debugger-interpreter has been announced by Microsoft. QuickBASIC boasts speeds up to 150,000 lines per minute, program modification and debugging without recompilation and on entry syntax checking. QuickBASIC 4 is a 'threaded p-code interpreter' which offers the speed of a compiler and the interactiveness of an interpreter. This technology incorporates users' changes to their programs at 150,000 lines per minute (Speed on an 8 Mhz PC-AT). Microsoft QuickBASIC 4 comes fully integrated with a subset of Microsoft's CodeView debugger and offers a multiple module programming support. Breakpoints can be set and cleared at any statement, 'watch' points can be used to conditionally break when expressions become true, and 'watch' expressions display the value of variables while the program is running. Expressions can be as a single variable or can refer to procedures written in BASIC or other Microsoft Languages, allowing programs to break on complex state changes. As well as mixed language support QuickBASIC 4.0 features Instant Help, EXE and library creation, and a Program Outliner, a function that allows the user to browse through an intended list of modules and procedures. Circle No 762

Ada Graphical Design And Development

A new software tool that allows software developers to build better applications by supplying a graphical interfacing technique was recently announced by Dublin based GENERICS Software Ltd. The product AnimAID, combines object oriented applications building blocks with a graphical interface generation tool. The product has been designed to allow custom tailoring of the applications interface to meet with customer requirements, whilst at the same time cutting development time. The system was developed as a result of Generics Software's work as the prime contractor in the ESPRIT 496 Papillon project, where the overall goal is to build a configurable graphics sub-system which can be integrated into existing and evolving systems. The User Interface Management component of the system is driven by an object-oriented definition of the application domain. AnimAID will be available on PCs next year at £2,500 and on Sun and Apollo Workstations at £6,500. Look out for a future article.

Circle No 763

MetaWare Compiler Support For The Weitek 1167

Compiler support for the Weitek 1167 floating point co-processor in both real and protected mode, has been announced by Metaware to coincide with Compaq's recent announcement of its new 80386 machines containing the Weitek 1167 numeric co-processor. Support is provided for both MetaWare's High C and Professional Pascal Languages. Support consists of a compiler enhanced to generate 1167 object code and, on non-UNIX systems, a run-time library that uses the 1167 for library math functions (such as sin and cos) and for conversions of floating point to ASCII and back. The compilers are available today. They have been successfully used by Weitek and Compaq to compile demonstration programs that illustrate how the 1167 can offer a speed-up of 1.5 to 2.8 over an Intel 80387 numeric coprocessor running at the same clock speed.

Circle No 764

PL/M For VAX

A PL/M cross compiler which produces target code for the 8051 family of microcontrollers, has been launched by Boston Systems Office. The package runs on VAX/VMS systems allowing a higher level of productivity. The compiler is said to produce more efficient code than compilers running on other systems. The BSO/PLM8051 compiler has all the standard PL/M 51 facilities that give programmers access to the features of the 8051, such as indirect addressing, bit manipulation and direct I/O or optimum use of the microcontroller's functions. BSO's full toolkit for the 8051 consists of a PLM compiler, assembler, linkage editor and symbolic software debugger. This complete package enables a developer to compile, assemble, simulate, debug and test the software before integrating it with the hardware.

Circle No 765

RBase and DBase III Compilers

Two new compilers, one for dBase III and the other for R:BASE have been imported from the states by In Touch. Force III, the dBase III compiler outputs stand alone code which is linkable through the standard DOS linker. As well as supporting all standard dBase commands, Force III features arrays, for next loops, sound, extensive maths capabilities, file primitives like FOPEN and FREAD and new numeric datatypes. The compiler can Handle up to 2.1 billion records per file, 2048 fields per record, 254 bytes per field, 10 DBF and 7 NDX files open simultaneously. FORCE III comes with complete documentation and the Miriam Liskin book 'Advanced dBase III Plus Programming and techniques'. Force III is priced at £129. R:Turbo claimed to be 'the first true compiler' for R:BASE allows the creation of standalone .EXE files. The user may define functions, use DOS batch files from within the code, and call C and ASM routines. R:Turbo also supports overlay structure and EMS allowing the application to exceed 640K.

Circle No 766

SkyBase 4

SkyBase V.4 has been launched by Sky Software. The new applications development system is a multi-user implementation running on DOS Networks and Xenix systems, and is designed to run on PCs and compatibles. SkyBase is aimed at the professional software developer designed to be the ideal tool for development of complete business applications or front end programs. New-comers to SkyBase 4 can be up and running with the system straight away due to the products Cobol-like syntax.

Circle No 767

CASE Environment for VAXStation 3200 And 3500

A set of Computer Aided Software Engineering tools (Cadre's Teamwork) running on DEC's new VAXStation 3200 and 3500 systems are now being distributed in the UK by Instrumatic. Teamwork can now benefit from a two to three times improvement in overall system performance without any code modifications. Cadre's family of computer-aided software engineering tools includes support for the automation of systems analysis (Teamwork/SA), real-time systems analysis (Teamwork/RT), information modelling (Teamwork/IM), and systems design (Teamwork/SD) phases of the software development lifecycle. In addition, Teamwork/ACCESS, a database and integration tool, provides users with the ability to integrate other VAX software development tools with Cadre's Teamwork product line to create a full lifecycle development environment. The Teamwork product line is currently available for the VAXStation 3200 and 3500 systems.

Circle No 768

New Case Integration Tool

Software Generation International Ltd has introduced the APS/PC Excelerator Integrator, which links Index Technology Corporation's analysis and design product, Excelerator, with the APS Development Centre. The APS/PC Excelerator enables users to transfer screens, reports and record descriptions created with Excelerator into the APS Development Centre. With the APS/PC Excelerator Integrator, users access menus within Excelerator to transfer design information from its dictionary to the APS Application Dictionary. The product is available immediately and is priced at £4,000 per site. It is an integrated software product that supports the system analysis and design process. It Includes a graphics facility for developing and revising system-related diagrams, an integrated Dictionary that stores, validates, and cross references design data and an integrated facility for the mock-up and prototyping of screens and reports. Excelerator runs on the the IBM PC XT, PC AT, a wide range of compatibles with at least 640K of RAM, and on DEC Vax station Line.

Circle No 769

Uniplus+ Development System

The development system, Accell, is now available under Uniplus+, Root's implementation of UNIX. The system provides developers with a development environment for multi-user business applications software and includes: a fourth generation language; an application generator; and a database based on Unify. Accell consists of five modules: Accell/DBS, Accell/Generator, Accell/Language, Accell/Environment and Accell/Manager. Accell/DBS includes all the features of Unify including a data dictionary and multiple access methods. Accell/Generator, this is an easy to use visual development tool for generating and prototyping an application. Accell/Language is a very powerful, non-procedural 4GL which also provides a 3GL interface and integrates SQL.

Circle No 770

CASE Tools For VAX Family

The Case Division of Tektronix UK Ltd's Design Automation Group has announced that it will support DEC's new MicroVAX 3000 series of minicomputers and workstations, by offering its full range of software development tools for use with the products. The Tektronix products to be offered on the MicroVAX 3000 include the company's Structured Analysis and structured Design Tools provide a graphical implementation of the technique outlined by De Marco in 'Structured Analysis and System Specification', together with all of its more recent extensions, such as those enabling the technique to be used for the analysis of real-time system. The company's LANDS environment include the Ada, Pascal and C languages.

Circle No 771

New On-Line Development Tools

A new development architecture and window-based application tools for building and running on-line, transaction-oriented applications have been announced by Sybase. The tools and architecture are provided in the Sybase System, a high-performance relational database management system for VAX/VMS and Sun/UNIX computers. The Sybase system is based on a requestor/server architecture where application functions can be handled separately from data management functions in a networked environment. Another important feature is that data integrity is controlled in the database manager rather than in the application. The Sybase Datatoolset includes window-based tools for building, running and maintaining applications on either character terminals or bit-mapped workstations. The DataToolset makes use of an overlapping windows, menus and a point-and-pick style user interface.

Circle No 772

PDOS Version 3.3

PDOS 3.3 is the result of enhancements made to the PDOS operating system by the Eyring Research Institute. Available in Britain from Andis Ltd, the operating system incorporates a new debugger, thereby increasing the functionality and versatility of the PDOS system, whilst the addition of new support products to the line of software development tools further increases programmer productivity. Under PDOS 3.3, event services have been expanded, allowing almost limitless event flags. These may reside totally within a task, alternatively within a group of tasks or across multiple processors. Multiple RAMdisks now form an integral part of the PDOS file management module. Previously a dynamically defined single RAMdisk was specified under the monitor or user program with multiple RAMdisks supported under the BIOS implementation. These were of a fixed size and required system generation to administer, but is not necessary with v.3.3.

Circle No 773

SPARC And System V

Plans for a new computing platform have been unveiled by AT&T and Sun Microsystem. The new platform will use a unified version of AT&T's System V, as well Sun's recently announced Scalable Processor Architecture (SPARC), a flexible microprocessor design for chips that use reduced instruction-set Architectures. It will include a standard interface, known as an application binary interface, or ABI, which will run UNIX system software programs as interchangeably as MS-DOS machines run PC software today. UNIX System V for the new platform will incorporate popular features of the Berkeley 4.2 system, a derivative of the UNIX system used widely in scientific and engineering markets, as well as features of SunOS, a variant of the Berkeley 4.2 system marketed by Sun. Features include networking and graphics such as the Network File System (NFS) and X.11/News, graphic user interfaces.

Circle No 774

Concurrent DOS 386 V.2.0

A new release of concurrent DOS 386 has been announced by Digital Research. Developed in Britain, Concurrent DOS 386 V.2.0 is a full featured, multi-user, multitasking operating system specifically designed to take advantage of the Intel 80386 microprocessor. It is now capable of running multiple copies of the most popular PC/MS-DOS applications on both the main personal computer console and on serial terminals. The new release of concurrent DOS 386 provides full DOS 3.3 compatibility and supports PC-DOS/MS-DOS 1.x, 2.x, 3.x applications on all consoles, as well as supporting memory resident programs such as Sidekick. Networked applications also run on the system giving the users full networking capabilities. Released concurrently is DOS XM V.6.0, a sister product targeted at 8086 and 80286-based micros. This has also been extended to to the DOS 3.3 level, and will run 'well-behaved' DOS applications on serial terminals. Pricing for the products remains the same at £395 for Concurrent DOS 386 and £295 for DOS XM.

Circle No 775

Multi-User PC-DOS

A new multi-user version of the Power System development environment for the IBM AT, 80386 computers plus compatibles and PS/2 family of computers has been released by Pecan for R&D applications. Although the Poly Power System has supported multi-tasking for some time, the new system makes it possible for developers to easily write multi-user applications. Up to three users are supported on a standard AT-class machine, with one user utilising the AT screen and the other users utilising inexpensive terminals. The development environment itself as well as applications using it, can also function in multi-user mode, so that up to three programmers can develop simultaneously on the same machine. A 'virtual terminal' mode is also supported, enabling the programmer or application user to switch between a maximum of three virtual terminals on the same workstation using a predefined hot key. The multi-user Poly Power System functions under DOS and, like the standard DOS power system, can be used to write DOS applications startable from DOS. Any of the Pecan languages, including Modula-2, can be used to write multi-user applications. The product is available immediately for £399.00 including a compiler.

Circle No 776

MS-OS/2 For Ericsson PCs

The Microsoft OS/2 operating system will soon be available on the range of Ericsson's PC compatibles, due to a licensing agreement with Microsoft. Microsoft's Windows Presentation Manager, which runs under OS/2 will provide users with a full graphics-based windowing system. The New PCs running OS/2 will be available from mid 1988 on the companies range of 80286 and 80386 based machines.

Circle No 777

New Advisor Expert System

Advisor-2 just launched by Expert Systems International, is an easy to use, expert system shell. Advisor-2 is a completely rewritten relative of ESP Advisor. The system is menu driven and is built on a colour-coded windowing system. Knowledge bases of up to 1 Megabyte can actually be chained together, so there is effectively no limit to the ultimate size of the system. One of the major features of Advisor-2 is its interfacing capability. Advisor-2 comes with interfaces to lotus 1-2-3, dBase, Lattice C, Microsoft languages, and ESI's Prolog-2, enabling users to link their own expert systems applications with existing programs written with traditional software packages. Advisor-2 costs £495, and comes with comprehensive documentation and a tutorial disk. It requires a minimum of 640K of RAM and runs on PCs and compatibles.

Circle No 778

Cache Memory Controller For The 80386

A cache memory controller for iAPX386 has been launched by Intel. The chip called the 82385, is now in volume production, with some of the first units going into the new 20MHz 80386 machine from Compaq. The 82385 Cache Controller contributes to a gain in system performance by eliminating processor wait states and by reducing bus accesses to main memory. It thus enables the potential performance of the 80386 to be fully realised and increases overall throughput by improving system bus utilisation. The 82385 provides advanced 32-bit cache control features, including a 'posted write' policy, which eliminates wait states when writing to main memory, and a bus watching mechanism that ensures cache coherency without any performance penalty. The 82385's interface to the 80386 microprocessor and to the system bus is software transparent. Thus, no software modification or new software is required to use the 82385 cache controller. The chip is available in bulk quantities of 10,000 and are priced at £89.38 per unit.

Circle No 779

10 MIPS Colour Workstation

Whitechapel Workstations, has introduced a major new range of UNIX colour workstations that are the first in the new generation of RISC-based workstations providing 10 MIPS sustained performance for less that £20,000. By including the MIPS Computer Systems R2000 Chip-set with integral floating point accelerators the HITECH-10 series provides a desktop graphics platform offering ten times the power of a VAX 11/780. Supporting UNIX 4.3bsd with System V extensions and the BRL SVID library, the workstations include both industry standard distributed window management systems: X-Windows (V11) and NeWS. The range can be incorporated efficiently onto existing heterogeneous networks via the Ethernet link. The workstations also provide an IBM PC AT bus interface to allow VARs and system builders to cost-effectively incorporate a variety of standard add-on boards into their applications such as modems and frame grabbers. Optimising compilers are available for C, Pascal and Fortran that significantly enhance CPU performance through innovations such as rearranging instruction flow to maximise the concurrency of operations, and by optimisation of floating point performance through scheduling the execution of instructions in parallel with general-purpose instructions. The workstations are available with either 95, 170 or 320Mb hard disks with an MS-DOS compatible floppy disk Drive.

Circle No 780

Please address all new product announcements to the News Editor, .EXE Magazine, 10 Barley Mow Passage, Chiswick, London W4 4PH.

T·R·A·I·N·I·N·G S·C·H·E·D·U·L·E

4GLs/Databases

01 Dec	Designing Data Base Systems	MSS	4 Days	£440
01 Dec	Informix-SQL	Sphinx	3 Days	£395
01 Dec	Intermediate Dataflex	Dflex	3 Days	£500
01 Dec	Revelation Level I	ICS	1 Day	£150
02 Dec	4th Generation Environment: T & P	LBMS	3 Days	£480
02 Dec	DB2 Data Base Design	F&S	3 Days	£655
02 Dec	Introductory Dataflex	Dflex	3 Days	£500
02 Dec	Revelation Level II	ICS	3 Day	£600
04 Dec	Informix Overview	Sphinx	1 Day	£125
07 Dec	Informix-4GL	Sphinx	3 Days	£395
07 Dec	Intermediate Dataflex	Dflex	3 Days	£500
07 Dec	Vista Programming 3	Vista	5 Days	£550
08 Dec	Sculptor For Programmers	MPD	2 Days	£250
09 Dec	Data Base Structures And Access	F&S	3 Days	£655
09 Dec	Introductory Dataflex	Dflex	3 Days	£500
14 Dec	Ad.Lib Workshop	Vista	2 Days	£250
14 Dec	Converting Informix 3.3 To SQL	InSet	3 Days	£475
14 Dec	Databases In Dist. Processing	F&S	3 Days	£655
14 Dec	Revelation Level III	ICS	2 Days	£500
14 Dec	Using dBase III and dBase III+	Dsolve	2 Days	£290
16 Dec	Informix-SQL	Oliv	1 Day	£140
16 Dec	Introductory Dataflex	Dflex	3 Days	£500
16 Dec	Rel. DBMS Theory & Practice	LBMS	2 Days	£360
16 Dec	Working With dBase	Digitus	1 Day	£155
17 Dec	Informix-4GL	Oliv	2 Days	£280
17 Dec	Programming with dBase III+	Digitus	2 Days	£295
04 Jan	Programming with Informix-SQL	Sphinx	3 Days	£395
11 Jan	4th Generation Languages	MEDC	4 Days	£275
11 Jan	Practical Dataflex	BEM	4 Days	£500
12 Jan	Relational Database Systems	ICSpub	4 Days	£745
12 Jan	Relational Databases: T & P	LBMS	2 Days	£360
18 Jan	Programming with Informix - 4GL	Sphinx	3 Days	£395
18 Jan	Unify Database - Introduction	Root	2 Days	£290
19 Jan	dBase III for Programmers	MEDC	4 Days	£275
19 Jan	Using dBase III+	MCU	2 Days	£350
21 Jan	Programming in dBase III+	MCU	2 Days	£350
25 Jan	Database Development Workshop	LBMS	5 Days	£960
27 Jan	Uniplex II+ Database	ITL	2 Days	£275

Analysis/Design

02 Dec	LSDM Overview	LBMS	2 Days	£360
07 Dec	Basic Business Syst. Analysis	BIS	5 Days	£835
07 Dec	Basic Syst. Analysis & Design	MSS	5 Days	£525
07 Dec	Business Syst. Design	Hosk	5 Days	£585
07 Dec	EPOS Fundamentals	Sphinx	5 Days	£750
07 Dec	Practical Syst. Design	BIS	5 Days	£835
07 Dec	Struct. Analysis Workshop	Your	5 Days	£695
07 Dec	Struct. Syst. Design	NCC	5 Days	£690
07 Dec	Struct. Syst. Design And Prog.	MSS	5 Days	£545
07 Dec	Struct. Syst. Design LSDM II	LBMS	5 Days	£960
08 Dec	S/w Engineering & Struct. Design	MCU	4 Days	£650
08 Dec	Struct. Design & Programming	ICSpub	4 Days	£745
09 Dec	Gecomco Users	GEC	2 Days	£330
09 Dec	Practical Sizing Using SSADM	BIS	3 Days	£495
11 Dec	Struct. Syst. Analysis LSDM I	LBMS	5 Days	£960
14 Dec	Basic Business Syst. Analysis	BIS	5 Days	£835
14 Dec	Syst. Design & Implementation	MSS	5 Days	£525
14 Dec	Computer Syst. Design	Hosk	5 Days	£585
14 Dec	Data Analysis Workshop	BIS	5 Days	£675
14 Dec	Struct. Syst. Design	BIS	5 Days	£675
14 Dec	Syst. Implementation	Hosk	5 Days	£585
15 Dec	Design of Real-Time Syst.	MSS	4 Days	£465
04 Jan	LSDM Part I	LBMS	5 Days	£960
04 Jan	Quality in Syst. Analysis & Design	MSS	2 Days	£320
04 Jan	Structured Syst. Analysis	BIS	5 Days	£835
04 Jan	Syst. Planning	Hosk	5 Days	£640
11 Jan	Appr. of Syst. Analysis & Design	MSS	2 Days	£320
11 Jan	Basic Business Syst. Analysis	BIS	5 Days	£835
11 Jan	Basic Syst. Analysis & Design	BIS	20 Days	£2900
11 Jan	Basic Syst. Analysis & Design	LBMS	4 Weeks	£3260
11 Jan	Business Syst. Analysis	Hosk	5 Days	£640
11 Jan	Business Syst. Analysis	MSS	5 Days	£560
11 Jan	LSDM for Programmers	LBMS	5 Days	£960
11 Jan	LSDM Part II	LBMS	5 Days	£960
11 Jan	Practical Analysis using SSADM	BIS	5 Days	£835
11 Jan	Practical Syst. Design	BIS	5 Days	£835
11 Jan	SSADM Part I	LBMS	5 Days	£775
12 Jan	Structured Design & Programming	ICSpub	4 Days	£745
18 Jan	Basic Business Syst. Analysis	BIS	5 Days	£835
18 Jan	Business Syst. Design	Hosk	5 Days	£640
18 Jan	Computer Syst. Design	Hosk	5 Days	£640
18 Jan	LSDM Part I	LBMS	5 Days	£960
18 Jan	Practical Syst. Design	BIS	5 Days	£835
18 Jan	Structured Syst. Analysis	BIS	5 Days	£835
25 Jan	LSDM Part II	LBMS	5 Days	£960
25 Jan	SSADM Part II	LBMS	5 Days	£775
25 Jan	Structured Analysis and Design	Hosk	5 Days	£640
25 Jan	Structured Programming Workshop	BIS	5 Days	£675
25 Jan	Structured Syst. Design	BIS	5 Days	£675
25 Jan	Syst. and Programming Supervision	BIS	5 Days	£835
25 Jan	Syst. Implementation	Hosk	5 Days	£640
26 Jan	Adv. Microprocessor System Design	ICSpub	4 Days	£745
26 Jan	Expert Syst. Design & Development	ICSpub	4 Days	£795
28 Jan	LSDM Overview	LBMS	2 Days	£360

UNIX/XENIX

01 Dec	Introduction To SCO XENIX	SCO	2 Days	£300
03 Dec	SCO XENIX System Administration	SCO	2 Days	£300
03 Dec	UNIX System Administration	Root	2 Days	£290
07 Dec	Device Drivers And Kernel Overview	InSet	5 Days	£895
07 Dec	Introduction To UNIX	Thom	3 Days	£330
07 Dec	UNIX Fundamentals I	InSet	5 Days	£625
07 Dec	XENIX For Support Staff	Oliv	5 Days	£700
09 Dec	Supporting SCO XENIX	SCO	3 Days	£525
14 Dec	Introduction To UNIX	Eq	3 Days	£450
14 Dec	UNIX - A Practical Course	Root	3 Days	£420
14 Dec	UNIX Fundamentals I	InSet	5 Days	£625
14 Dec	UNIX V.3 Kernel Workshop	InSet	5 Days	£1250
14 Dec	UNIX/XENIX Fundamentals	Sphinx	3 Days	£395
15 Dec	Programming For Networked XENIX	SCO	3 Days	£600
15 Dec	UNIX Hands-On Workshop	ICSpub	4 Days	£745
17 Dec	UNIX - An Overview	Root	1 Day	£150
17 Dec	UNIX For System Managers	Eq	2 Days	£300
17 Dec	UNIX System Managers	Digitus	2 Days	£295
17 Dec	XENIX For The PC-AT	Sphinx	1 Day	£125
04 Jan	UNIX Operating System	NCR	5 Days	£670
06 Jan	UNIX Syst. Programming	Root	3 Days	£420
11 Jan	32000 Intro to UNIX	NatS	2 Days	£POA
11 Jan	Setting Up UNIX Systems	ITL	2 Days	£285
11 Jan	UNIX System Administration	NCR	3 Days	£402
12 Jan	UNIX	MCU	4 Days	£685
13 Jan	UNIX Systems Administration	ITL	1 Day	£145
25 Jan	UNIX OS Fundamentals		3 Days	£395

C Programming

01 Dec	Hands-On Programming In C	ICSpub	4 Days	£745
07 Dec	Advanced C Software Development	QA	3 Days	£420
07 Dec	Advanced C Workshop	InSet	5 Days	£725
07 Dec	C Programming	Root	5 Days	£650
07 Dec	C Programming Workshop	InSet	5 Days	£625
07 Dec	The C Programming Language	ConV	5 Days	£600
08 Dec	Programming In C	MCU	4 Days	£650
10 Dec	C Programming Standards	QA	1 Day	£180
10 Dec	C++: A Technical Overview	InSet	1 Day	£195
14 Dec	C Programming Under DOS	QA	5 Days	£580
14 Dec	C Programming Workshop	InSet	5 Days	£625
14 Dec	Introduction To C	Thom	3 Days	£330
15 Dec	Basic C Programming	MSS	4 Days	£465
11 Jan	C Programming	Root	5 Days	£650
11 Jan	C Programming	Sphinx	5 Days	£595
12 Jan	Advanced Programming in C	ICSpub	4 Days	£795
13 Jan	C Programming	NatS	3 Days	£POA
18 Jan	C Programming	Intel	5 Days	£700
26 Jan	Programming in C	MCU	4 Days	£685

Ada/Pascal/Modula-2

02 Dec	Basic Pascal Programming	MSS	3 Days	£390
07 Dec	Ada Programming Course	AdaT	5 Days	£600
14 Dec	Ada Programming Course	AdaT	5 Days	£600
14 Dec	Getting Results from Modula-2	RTA	3 Days	£550
15 Dec	Ada: A Managers Overview	InSet	1 Day	£725
11 Jan	Ada Programming Course	AdaT	5 Days	£770

T·R·A·I·N·I·N·G S·C·H·E·D·U·L·E

19 Jan	Ada Management Seminar	AdaT	1 Day	£132
20 Jan	Ada for Technical Management	AdaT	2 Days	£275
25 Jan	Ada Programming Course	AdaT	5 Days	£770
25 Jan	Advanced Ada Workshop	AdaT	5 Days	£880
26 Jan	Ada Prog. & Software Engineering	ICSpub	4 Days	£745

Other Languages

01 Dec	RPGII Prog For Experienced	MSS	3 Days	£390
02 Dec	Programming In BASIC	MSS	3 Days	£390
07 Dec	RPG III System/38 Programming	BIS	2 Days	£370
08 Dec	APL Fundamentals	C&D	3 Days	£445
14 Dec	Basic PL/1 Programming	MSS	5 Days	£525
14 Dec	Occam Programming Workshop	InSet	5 Days	£625
04 Jan	Basic FORTRAN Programming	MSS	5 Days	£560
04 Jan	COBOL Programming	MSS	3 Days	£410
04 Jan	PL/M 51 Programming	Intel	4 Days	£700
04 Jan	PL/M Programming	Intel	5 Days	£700
11 Jan	Basic COBOL Programming	MSS	5 Days	£560
18 Jan	Basic RPG11 Programming	MSS	5 Days	£560
25 Jan	Structured COBOL Programming	MSS	3 Days	£410

Processor Programming

07 Dec	MC68000	Mote	4 Days	£650
07 Dec	Series 32000 Development	NS	3 Days	£POA
14 Dec	8088/86/286 PC Assembler Apps.	QA	5 Days	£580
14 Dec	MC68020	Mote	4 Days	£650
14 Dec	Using VersaDOS	Mote	4 Days	£650
15 Dec	Microprocessor S/w H/w Interf.	ICSpub	4 Days	£745
04 Jan	8086 & 80286 Real Mode Prog.	Intel	5 Days	£650
11 Jan	80386 System Software Workshop	Intel	5 Days	£750
12 Jan	8086 Assembler and The PC	MCU	4 Days	£685
18 Jan	32000 Instruction Set	NatS	4 Days	£POA
25 Jan	32000 Development Environment	NatS	3 Days	£POA
25 Jan	Programming using ASM 386	Intel	5 Days	£750
26 Jan	Programming and Int. the MC68000	MCU	4 Days	£685

Comms and Graphics Programming

01 Dec	Distributed Processing	MSS	2 Days	£300
01 Dec	Graphics Using GKS/VDI Tools	ICSpub	4 Days	£795
01 Dec	Understanding Local Area Networks	ITL	1 Day	£175
02 Dec	Data Networks	BIS	3 Days	£495
02 Dec	Open Systems Interconnect (OSI)	ITL	1 Day	£175
07 Dec	Developing On-Line Applications	BIS	5 Days	£675
08 Dec	PC Networks Programming	QA	4 Days	£620
08 Dec	The DoD TCP/IP Protocol	InSet	1 Day	£245
15 Dec	Micro-To-Mainframe Linking	Clar	2 Days	£449
16 Dec	Token Ring Technology	ITL	2 Days	£375
13 Jan	Evaluating and Implementing LANs	F&S	3 Days	£655
18 Jan	Understanding Networks	ITL	1 Day	£175
19 Jan	Understanding Local Area Networks	ITL	1 Day	£175
20 Jan	Open Systems Interconnect	ITL	1 Day	£175
20 Jan	The OSI Reference Model	F&S	3 Days	£655

Programming Management

07 Dec	Software Project Management	NCC	2 Days	£345
08 Dec	Project Management Tool Set	GEC	1 Day	£200
08 Dec	Software Rel. & Quality Control	MCU	4 Days	£650
14 Dec	G-Taskplan - Users	GEC	2 Days	£330
15 Dec	Software Req. Spec. & Test	ICSpub	4 Days	£745
15 Dec	Software Testing & Verification	MCU	4 Days	£650
04 Jan	Professional Programming	Hosk	5 Days	£640
05 Jan	Software Quality Assurance	MCU	4 Days	£685
19 Jan	How to Manage Software Projects	ICSpub	4 Days	£795
19 Jan	Practical Project Management	BIS	3 Days	£675
19 Jan	Soft. Engineering & Struc. Design	MCU	4 Days	£685
25 Jan	Project Planning and Control	BIS	3 Days	£495

Prolog/Lisp/AI

01 Dec	Intermediate Prolog	ESI	2 Days	£330
03 Dec	Advanced Prolog	ESI	2 Days	£330
07 Dec	Knowledge Engineering	F&S	3 Days	£655
09 Dec	Intro To Expert Systems And Prolog	Tele	1 Day	£125
14 Dec	Expert System Project Management	ESI	1 Day	£180
15 Dec	Introduction To Expert Systems	ESI	2 Days	£330

PC Operating Environments

01 Dec	Introduction To DOS	Digitus	1 Day	£135
01 Dec	MS-OS/2 Application Programming	QA	4 Days	£760
08 Dec	Intro to 8088/86/286 Based PC's	QA	4 Days	£520
08 Dec	MS Windows App Programming	QA	4 Days	£680
15 Dec	MS-OS/2 Application Programming	QA	4 Days	£760
21 Dec	Introduction To DOS	Digitus	1 Day	£135
27 Jan	Interfacing to DOS and the BIOS	MCU	3 Days	£575
12 Jan	Introduction to DOS	1 NCR	1 Day	£120

Other Operating Systems

07 Dec	UniTECS Application Development	Root	5 Days	£750
11 Jan	iRMX 286 Operating System	Intel	3 Days	£POA
13 Jan	Real Time Software & Systems	MCU	3 Days	£575

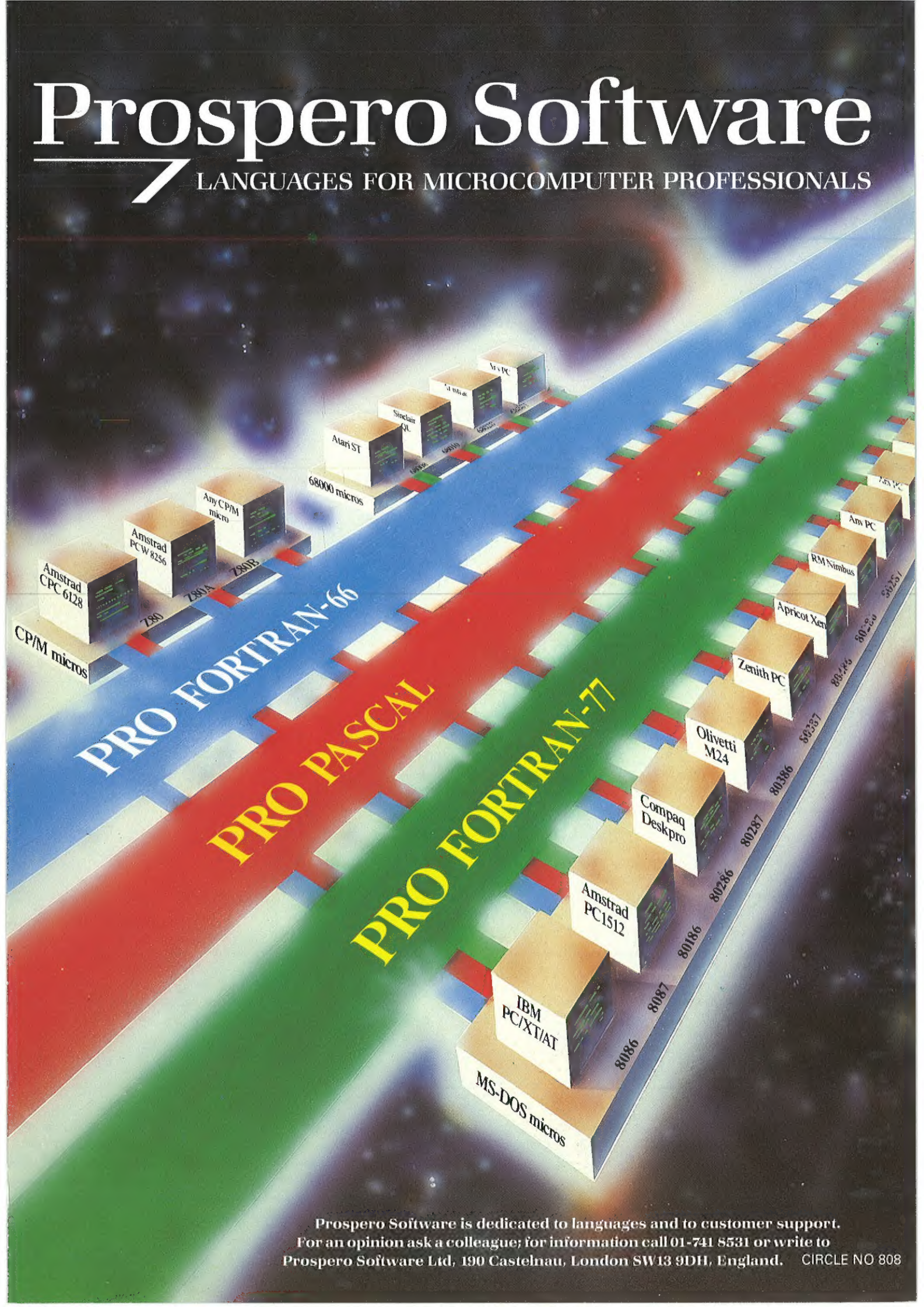
ABS Computers	0273 421509
AdaT (Ada Training Ltd)	0279 725030
Alsys	0491 579090
APL Ltd	0244 46024
AT&T Unix Europe Ltd	01 567 7711
BEM	0328 700494
BIC Systems	0232 233278
BIS	01 633 0866
BOC	01 741 9345
BOS	01 430 2438
CCTC	05435 2511
Clar (Clarion)	0306 884732
ConV (Convergent Tech)	0344 411707
Data Logic (Dlog)	091 863 0383
Dataflex (Dflex)	01 729 4460
Digitus	01 379 6968
Dsolve (Datasolve)	01 828 7878
Eq (Equinox)	01 729 4460
ESI (Expert Systems Int)	0865 242206
F&S (Frost & Sullivan)	01 730 3438
GEC Software	01 240 7171
HIS (High Integrity Systems)	0279 725030
Hosk (Hoskyns Group)	01 434 2171
ICS (Intelligent Comp Svs)	0256 469460
ICSpub (ICS Publishing)	0372 379211
InSet (Instruction Set)	01 251 2128
Intel	0793 696000
Istel Ltd	01 831 0361
ITL (Information Tech)	01 839 1028
LBMS Ltd	01 636 4213
LPA (Logic Programming Ass)	01 871 2016
MCU (Microcomputer Unit)	01 405 3020
MEDC	041 887 0962
Mercia Software Ltd	021 359 5096
Mote (Motorola)	0296 395252
MPD	01 328 2277
MSS Services Ltd	0903 34755/6
Mtec (Microtec Research)	0256 57551
NCC (National Computing Center)	061 228 6333
NCR	021 742 7731
NS (National Semiconductor)	010 49 08141 103486
Oliv (British Olivetti)	0428 4011
P&B (Porter Burns Inc)	01 328 2712
Pro-Lab Plc	0223 323151
QA Training	0285 5888
Root Training	01 606 7799
RTA	01 656 7333/4/5
SCO Training	01 637 9111
Sperry Ltd	021 632 4171
Sphinx Ltd	0628 75343
Symb (Symbolics)	0494 443711
Syst (Systematica SE)	0202 297292
Tele (Telecomputing)	0865 777755
Thom (Thomson Computers)	0904 611666
Unisys	0908 665211
VISTA Computer Systems	01 434 9801
Your (Yourdon)	01 637 2182

/shutdown

As Jeremy Thomas of **Root** (who has just bought US-based **Unisoft**) prepares to implement a '**multi-part strategy**' to ship a lot of UNIX, **/shutdown** is pleased to report that despite the fast-approaching twilight of 1987, the industry's fervour to talk a lot of convoluted rubbish about itself remains undaunted. Perhaps it's this that's forcing the brain drain, as Root are followed by **Zorland** to set up US offices, going under the name of **Zortech**, presumably because Americans can't tell the difference between B and Z (pronounced *bee* and *zee*, though if you put them the other way around would presumably be *zed* and *bed*) and Zorland were loathe to be confused with anyone else. Letters, it seems, can also confuse the *European* market, as **Sphinx** announced a recent letter change amidst the pomp normally afforded to major announcements. The 'U' in ICUS (Sphinx' software distribution network) is now to be an 'O', because **open** is a much nicer word than **UNIX**. And on the same note, **fat** is a much better word and thing to be than anything else, because NEC now has the computer for you, following their announcement of their superhigh performance PC with the wonderous (and utterly faithful transcript) phrase '**aimed at the heavier user**'. Needless to say, it allows for 'internal expansion', which is vaguely wierd (not to mention weightist), but not nearly so obscure as a recent Living C mailshot, which in its endeavours to attract prunters (programmers who are potential purchasers) offers a **Borland** (that's Borland with a B) Turbo C and Living C package, says 'programming in C is a bit like sculpting in bronze - a very hard metal'. Perhaps readers should switch to Modula-2, following another recent *let's jump on the Borland Zandwagon* announcement from **Real Time Associates** (who actually don't have a lot of time) offering a Modula-2 compiler 'which outperforms Turbo C'. While you look up the phone number for Rent-a-Bandwagon, ask yourself why a Modula-2 definition module is always lonely, or what language Italian UNIX programmers use (answers: a definition module has no body and Italians use 'si'), and relax for the monthly '**I Still Take Myself Far Too Seriously**' awards, which go to **Wang** who, following a mildly savage lambasting last month, persisted in bombarding the cosmos with rubbish about their WITS show. The latest offensive (a press release) cites a 'two-tiered approach' used to launch several new products and a demonstration of 'technologies such as voice and telephony'. Most voices aren't technology and 'telephony', well, even the combined braincell of the .EXE editorial team can't recall the use of this term in the last 80 years or so. And talking of years, as we end this one (and this **/shutdown**) we'd like to wish our readers a very Happy (if slightly premature), Jargon-Free and Bandwagonless Christmas. Zye zye!

Prospero Software

LANGUAGES FOR MICROCOMPUTER PROFESSIONALS



Prospero Software is dedicated to languages and to customer support.
For an opinion ask a colleague; for information call 01-741 8531 or write to
Prospero Software Ltd, 190 Castelnau, London SW13 9DH, England. CIRCLE NO 808

New Clipper™ Breaks the Speed Barrier

Since the launch of Clipper, dBase applications have reached new heights.
Making them faster, free from royalty fees and runtime modules and with Clipper there's no need for a LAN pack, because Autumn '86 brought you networking capabilities.
At Nantucket we're never slow to make a product even better.
And with Clipper Summer '87 you'll be on cloud nine.
All the famous features are still here.
A huge new list of commands and functions, faster indexing, use of expanded memory, no copy protection, and the ability to run applications on major networks.
But now it really flies.
So quick in fact, that in popular benchmarks, Summer '87 leaves the competition grounded.
It now sports a Compatible index system too.

Allowing programmers to save time setting up compiled programs due to direct access to dBase indexes.
Summer '87 also allows 250 files to be open at once. Which means you can access information quicker and write your programs with even greater manouverability. (This facility requires DOS 3.3).
Add to this a near miss search facility, additional memo field functions, extra array handling and you have the perfect dBase compiler, that's also compatible with Microsoft C.
Perhaps the best news of all is that Autumn '86 users can update to Summer '87 absolutely free. And as Nantucket have offices in the U.K. we're able to provide direct customer support.
So if you and your colleagues would like even more networking power you can phone us today for further details.
And watch your dBase capability soar.

New Clipper™ – does it quicker

CIRCLE NO 809

Nantucket®

NANTUCKET CORPORATION EUROPE, BLUECOATS AVENUE, FORE STREET
HERTFORD, HERTS SG14 1PB. TELEPHONE 0992 554621.
TELEX 265871 MONREF G REF NNE 077. FAX 0992 554934.